Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES:
SONIDO E IMAGEN

AUTOMATIZACIÓN DE INSTALACIÓN Y CONFIGURACIÓN DE APLICACIONES PEER-TO-PEER EN UN ENTORNO VIRTUALIZADO BASADO EN MODELNET Y PROXMOX

Autor: Félix Gómez Fernández

Tutor: Isaías Martínez Yelmo

Leganés, Diciembre de 2010

Título: **AUTOMATIZACIÓN DE INSTALACIÓN Y CONFIGURACIÓN DE APLICACIONES PEER-TO-PEER EN UN ENTORNO VIRTUALIZADO BASADO EN MODELNET Y PROXMOX**

Autor: Félix Gómez Fernández

EL TRIBUNAL

Presidente: Carmen Guerrero López

Vocal: Eva Rajo Iglesias

Secretario: Julio Villena Román

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 16 de Diciembre de 2010 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo. PRESIDENTE Fdo. SECRETARIO Fdo. VOCAL

A mis padres y a las personas
que ya no están con nosotros

"Los grandes logros no se miden por los resultados, sino por los esfuerzos"
- Gandhi -

"Sólo lo difícil de alcanzar, tiene valor para guardar"
- Maclidel -

Agradecimientos

Al fin he llegado hasta este punto, el cual me ha sido verdaderamente muy difícil de alcanzar en todos los aspectos, económico, de motivación, de recursos y una larga lista de inconvenientes, pero ahora no es el momento de quejarse, si no de estar orgulloso de todo el esfuerzo realizado por mí y sobre todo de las personas que me han rodeado y apoyado en esta dura etapa universitaria.

En primer lugar quiero agradecer a mis padres Félix y Ángeles por darme TODO, y cuando digo todo, hablo de su apoyo moral y económico, su amor y todo su cariño en los momentos malos y mostrarme así el camino correcto para la consecución de mis objetivos como estudiante y sobre todo como persona. También quiero agradecer a mis hermanas Yolanda y Mª Ángeles por apoyarme moral, psicológica y logísticamente en toda esta andadura como estudiante y que espero poder devolverles cada minuto de su ayuda, que en todo momento fue sincera. Mencionar también a mi cuñado Chema por su apoyo en este PFC y en algunas otras cosillas mas durante la carrera y a Miguel que aunque ha llegado después, quiero agradecerle su apoyo como parte de la familia.

A Pilar, por soportar y aguantarme en los momentos de agobio y mal humor (por desgracia demasiados) y por animarme sin parar y hacerme sentir mejor cuando estaba con ella, olvidándome, aunque fuera por poco tiempo de la exigencia académica de esta universidad.

A mis amigos dentro de la universidad, Fran (Francis) y Alberto (Albertus), por todos esos buenos momentos en clases, prácticas y exámenes. Y sobre todo en esos viajes por la geografía española hospedándonos en pensiones la mar de extravagantes convirtiéndonos así en algo más que compañeros de universidad, en amigos. También a Pablo que en estos últimos años hemos compartido nuestra mayor pasión, el fútbol, dentro de la universidad y a todos ellos que en cualquier momento se han cruzado en mi vida universitaria como Javito, Sergio, Camarma, Clara, etc... Fuera de la universidad quiero hacer mención especial a Carlos por aceptarme y soportarme con mis paranoias de todo tipo alejándome así de los problemas de la universidad y también a mis amigos que por un motivo u otro ya no estamos juntos pero que en su día entendieron y aceptaron que tenía unas obligaciones como estudiante, las cuales respetaron.

Por último quiero agradecer a mi tutor Isaías, por llenarse de paciencia y hacer que un estudiante de Sonido e Imagen, con todo lo que eso conlleva, realice un proyecto de Telemática satisfactoriamente. También a Roberto por facilitarme todo tipo de información y recursos para que este trabajo sea lo mas llevadero posible, ya que los comienzos fueron muy difíciles.

Lo peor de estos casos es que siempre te olvidas de personas importantes sin querer, por eso quiero agradecer a todos los que directa o indirectamente me han ayudado y apoyado en esta etapa.

MUCHAS GRACIAS A TODOS.

Resumen

Desde hace unos años, el tráfico generado por el intercambio y transferencia de contenido multimedia en Internet usando arquitecturas distintas a las cliente-servidor ha ido en creciente aumento. De ahí, tecnologías como la P2P (peer-to-peer) están teniendo mucho éxito y por eso se está trabajando en su estudio, entendimiento y desarrollo de nuevas soluciones de manera muy activa.

Un tipo de estas arquitecturas son las redes overlay, que lo que hacen es crear una red sobre Internet con una topología lógica diferente a la del nivel de red. Dependiendo de la manera en que se genere serán estructuradas o no.

El principal objetivo de este proyecto es diseñar una herramienta que automatice todo el proceso previo a la realización de experimentos enfocados a la demostración, estudio y evaluación del rendimiento de redes overlay, dotando así a desarrolladores e investigadores con una aplicación que les ayudará a la obtención de resultados y conclusiones con mayor rapidez, facilidad y fiabilidad centrando su atención exclusivamente en el desarrollo de sus propios experimentos, siendo estos el paso anterior al despliegue en entornos reales como Internet.

Una vez desarrollada la herramienta, se ha procedido a la comprobación y validación de esta usando una red emulada creada por el software Modelnet, que nos permite crear clusters de emulación de una manera escalable y flexible.

Finalmente se procedió a la realización de unos experimentos nuevos con el objetivo de añadir más información al desarrollo del estudio en este ámbito. Por último, cabe destacar el uso de PROXMOX como el entorno de virtualización y monitorización de equipos físicos usado durante el proyecto para realizar experimentos, capaz de gestionar un número elevado de nodos virtuales, que en este caso fueron 1000.

Abstract

In recent years, the traffic generated by the exchange and transfer of multimedia content over the Internet using different architectures to client-server has been on the growing increase. Hence, technologies such as P2P (peer-to-peer) are having so much success and are working in his study, understanding and developing new solutions very active.

One type of these architectures are overlay networks that they do is create a network on the Internet with a different logical topology of the network level. Depending on how they generate will be structured or not.

The main objective of this project is to design a tool that automates the whole process prior to the completion of experiments focused on the demostration and evaluation study of the performance of overlay networks, thus providing developers and researchers with an application that will help them obtain results and conclusions more quickly, easily and reliably to focus exclusively on developing their own experiments, and these pre-deployment step in real environments as the Internet.

Once the tool has been developed process of verification and validation is using an emulated network created by the Modelnet software that lets you create clusters of emulation in a scalable and flexible.

Finally we proceeded to carry out new experiments in order to add more information to study development in this area. Finally, it should emphasize the use of Proxmox as the virtualization environment and monitoring of hardware used for the project to conduct experiments that can handle a large number of virtual nodes, which in this case was 1000.

Índice general

Índice de figuras

Índice de tablas

Capítulo I

INTRODUCCIÓN

1 Introducción

Las tecnologías y el uso de estas van cambiando según las necesidades del momento, y en estos años la transmisión de contenido multimedia se ha convertido en uno de los servicios más usados en Internet. Por ello, las necesidades van creciendo y estos nuevos servicios necesitan un amplio estudio para proveer más y mejores servicios.

El principal problema que comparten las tecnologías pioneras en la compartición de archivos como Napster, y las actuales aplicaciones de comunicación de video y voz en tiempo real como Skype, es que tienen que buscar la información en una red muy vasta, y en la cual los identificadores son las direcciones IP, que para más complicación no tienen ninguna relación con el contenido guardado en las máquinas que representan.

Localizado el problema, introducimos las redes overlay, cuya principal característica es que los terminales informáticos que la conforman se organizan de tal manera, que aún estando interconectados entre sí mediante routers IP, están organizados formando una nueva estructura de red, además de la existente según unos criterios establecidos. Tendremos por tanto, sistemas distribuidos, utilizados por ejemplo para encontrar contactos en un sistema de VoIP. Uno de los ejemplos más conocidos de este tipo de redes son las P2P, muy eficientes para la descarga de grandes volúmenes de información. Dentro de las redes overlay P2P distinguimos las estructuradas y las no estructuradas.

Las redes P2P no estructuradas tienen la peculiaridad de que se establecen arbitrariamente. Cada nodo se une a la red copiando los enlaces existentes de otro nodo y después forma sus propios enlaces en un plazo determinado. La desventaja de este tipo de redes es que si un usuario quiere encontrar una información en la red, su petición tiene que recorrerla toda, y se puede dar el caso de que dicha petición no se pueda resolver si el contenido no es muy popular. Esto sucede porque no hay correlación entre un usuario y el contenido compartido por él. Un ejemplo de estas redes son Napster o Gnutella.

Por el contrario, las redes P2P estructuradas superan estas limitaciones. Esto lo hacen manteniendo lo que se denomina tabla de hash distribuida (DHT), que permite que cada usuario sea responsable de una parte específica del contenido en la red. Posteriormente, siguen un protocolo global que determina qué usuario es responsable de qué contenido. De esta forma, cuando un usuario quiera buscar un tipo determinado de datos, usará el protocolo global para determinar que usuarios lo tiene y dirigir así la búsqueda hacia ellos. Algunos son Chord o Tulip Overlay.

El objetivo principal de este proyecto es, la realización de una herramienta que permita la automatización de todos los procesos para la construcción de una estructura virtual, en la cual se permita la realización de experimentos, con el fin de facilitar la labor de investigadores y desarrolladores para que centren sus esfuerzos en el diseño del tipo de experimento y no en el despliegue del escenario a evaluar.

Todo ello se realizará partiendo de, una modificación previa de una implementación de un protocolo de redes overlay estructuradas para que se comporte de un modo jerárquico,esperando mejoras en las prestaciones obtenidas.

Las fases del desarrollo del proyecto han sido básicamente las mismas que en cualquier Proyecto Fin de Carrera; es decir, como primer paso, la investigación de las herramientas a usar como Modelnet, Java, Linux, Netbeans, Proxmox, etc. En segundo lugar, el diseño e implementación de la herramienta existente para adecuarla a nuestros objetivos y por último, la fase de realización de pruebas para obtener los resultados pertinentes.

2 **Estructura de la memoria**

El presente documento se divide en los capítulos que se enumeran a continuación:

- **Capítulo I:** Breve INTRODUCCIÓN, enumerando los tipos de redes overlay y algunos ejemplos, tecnologías usadas durante el proyecto y objetivo principal del mismo. También se realiza una descripción del contenido de este documento.

- **Capítulo II:** Contiene el ESTADO DEL ARTE y describe detalladamente las tecnologías usadas, así como los diferentes tipos de redes que podemos encontrar.

- **Capítulo III:** Describe el proceso de DESARROLLO Y DISEÑO DE LA APLICACIÓN y el escenario desde el que se parte. Habla de los puntos débiles de la aplicación de partida y de la forma en la que se resuelve el problema para obtener la nueva solución.

- **Capítulo IV:** Forma de VALIDACIÓN de la nueva solución adoptada para cumplir los requisitos y objetivos para la realización del proyecto. Tras una breve introducción que ayudará a comprender los resultados esperados, se mostrarán las herramientas y configuraciones usadas para realizar las pruebas. Finalmente se muestran y analizan los resultados obtenidos.

- **Capítulo V:** Una vez validado el nuevo programa, se pasa a la GENERACIÓN DE NUEVOS RESULTADOS, los cuales otorgarán a los futuros investigadores una base a partir de la cual establecer unos criterios de estudio en este tipo de redes.

- **Capítulo VI:** Presenta las CONCLUSIONES obtenidas durante todo el proceso de realización de este proyecto y para concluir enumera varias líneas de trabajo futuras.

Capítulo II

ESTADO DEL ARTE

En este capítulo, se mostrará todo lo necesario para dar a conocer al lector lo que ha sido necesario para el desarrollo de este trabajo. En primer lugar, se explicarán las redes P2P, que son las que se usan en este proyecto. De ellas se verán las características que deben tener, cómo se las puede clasificar dependiendo del grado de centralización y estructuración y por último, las aplicaciones que se basan en este tipo de redes. Posteriormente, se comentará la aplicación desde la que se parte y las modificaciones que se hicieron en el proyecto de partida sobre el protocolo P2PP, que es el que se usa en ambos proyectos. A continuación, se explicará el emulador utilizado para poder lanzar los experimentos y poder así verificar el funcionamiento, este es ModelNet.

Para acabar este capítulo, se mostrarán los dos lenguajes de programación usados, las características que tienen individualmente y lo más importante para este proyecto, por qué y cómo pueden trabajar conjuntamente para extraer la información contenida en el archivo XML principal y realizar con dicha información las tareas de despliegue de la red en ModelNet, la instalación de los programas y archivos necesarios en todas las máquinas que van a participar en los experimentos, lanzar tales experimentos y por último, recoger todos los datos que se han ido creando en la ejecución para la extracción de resultados y su posterior evaluación.

1 P2P

1.1 Introducción

Los redes P2P (peer-to-peer) consisten esencialmente en sistemas de información distribuidos. La red gestiona los intercambios entre *peers* (usuarios) usando los recursos de sus propios ordenadores. Dependiendo del ámbito, se pueden clasificar los tipos de redes según su grado de centralización (centralizada, descentralizada o híbrida) o según su estructuración (estructurada y no estructurada).

1.2 Características

Una red P2P puede ser de distinta naturaleza, pero se le presuponen una serie de características, que la pueden hacer mejor para según que tipo de aplicaciones se vaya a usar. A continuación se detallarán las principales características.

1.2.1 Descentralización

Todos los nodos tienen la misma importancia, no existen nodos especiales y en consecuencia, ningún nodo es imprescindible para el funcionamiento de la red. Esto supone una ventaja respecto a la tolerancia de fallos, ineficiencias, recursos desperdiciados o cuellos de botella que se pueden producir en un servidor central.

Por otro lado, los inconvenientes, vienen a la hora de realizar las búsquedas y al controlar el derecho de acceso a los datos. Esto sucede porque ningún nodo tiene una visión global de los peers, ficheros o recursos disponibles en la red. Esta característica conlleva que, encontrar los recursos en la red sea mucho más complicado y lento, de ahí la necesidad de introducir la figura de supernodo o superpeer en el caso que fuera necesario. Una ventaja más que se puede destacar es que, debido a la falta de un servidor central, el coste de los recursos y de mantenimiento es mucho menor.

1.2.2 **Escalabilidad**

Esta característica es un beneficio inmediato de la descentralización, ya que, cuantos más nodos estén conectados a una red P2P, mejor será su funcionamiento. Cuando los nodos llegan y comparten sus propios recursos, automáticamente los recursos totales del sistema aumentan, y como consecuencia, se reducen las ineficiencias del mismo. De cualquier forma, la escalabilidad nunca debe conseguirse a costa de degradar otras características deseables, tales como el determinismo y el rendimiento. Este factor, está limitado por la cantidad de operaciones centralizadas que deben ser ejecutadas o la cantidad de estados que deben ser mantenidos o el modelo de programación empleado en el desarrollo de la aplicación.

1.2.3 **Rendimiento**

Esta característica también es una de las consecuencias de la descentralización. Con ella, se obtiene capacidad de almacenamiento y ciclos de computación distribuidos, así como un aprovechamiento mayor del ancho de banda. En las redes P2P, los sistemas basados en modelos híbridos son los mejores debido a la introducción de los supernodos y su función de agilizar las búsquedas.

Existen tres mecanismos para optimizar el rendimiento en este tipo de sistemas distribuidos, que son:

- **Replicación.** Consiste en poner copias de objetos más cerca de los nodos que solicitan determinado archivo (minimiza el tiempo de transferencia).

- **Almacenamiento en caché.** Reduce el número de mensajes entre nodos, con lo que se minimiza la latencia del acceso a los nodos y se maximiza la velocidad de las solicitudes. También balancea la carga de trabajo en la red.

- **Encaminamiento inteligente.** Trata de garantizar que los usuarios con intereses comunes estén lo más cerca posible dentro de la red.

1.2.4 **Robustez**

En los modelos cliente-servidor se tiene el problema que si falla el servidor no se puede acceder a los servicios que este provee. Sin embargo, en las redes descentralizadas, donde todos los nodos son iguales, no existe tal problema, ya que si falla un nodo no falla el sistema al completo. El único problema se producirá cuando un nodo falle y por tanto, no se podrá descargar nada de él, por eso la replicación de contenido puede aliviar este problema.

1.2.5 **Interoperabilidad**

La interoperabilidad es definida por la IEEE [IEE10] como *"la habilidad de dos o más sistemas o componentes para intercambiar información y para usar la información que ha sido intercambiada"*. Actualmente existen numerosas implementaciones de aplicaciones P2P y cada una con sus propias especificaciones, con lo que, aunque su propósito es el mismo, no son capaces de comunicarse entre sí.

Para solucionar el problema, se han formado, el grupo de trabajo de P2P de la IEEE y la JXTA [JXT10]. El primero intenta realizar especificaciones comunes para el entendimiento entre todas las aplicaciones y el segundo, es una plataforma creada por Sus Microsystems en 2001 que ofrece una infraestructura de código abierto para el desarrollo de dichas aplicaciones. Además también hay que destacar P2P-SIP de la IETF [IETF10], que es una implementación de un sistema de voz sobre el protocolo de Internet (VoIP) [VOIP10] con el que se pueden realizar llamadas usando una red P2P entre dos puntos finales con SIP [SIP10]. La IETF lo que quiere hacer con el grupo de trabajo P2PSIP es desarrollar unas especificaciones para el uso de la ubicación del recurso dentro de la red.

1.3 **Clasificación de redes según su grado de centralización**

1.3.1 **Centralizadas**

Son las redes P2P de primera generación que basan su funcionamiento en una estructura cliente-servidor. El servidor central tiene una base de datos con la información de los archivos almacenados por cada nodo y la actualiza cada vez que un cliente se conecta o desconecta de la red. Cuando un usuario solicita cierta información, el servidor le pone en contacto con el nodo que contiene los datos deseados, pero en ningún momento los contenidos son almacenados en dicho servidor.

El rendimiento a la hora de localizar recursos es muy elevado, ya que las búsquedas se realizan rápida y eficientemente, debido a que todos los nodos han de registrarse cuando acceden a la red. Sin embargo es un sistema muy costoso en lo que al servidor central se refiere si la red es grande y también tienen el problema de un punto único de fallo, siendo estas vulnerables. Este tipo de redes tienen ciertos problemas legales, véase el ejemplo de Napster [NAP10].

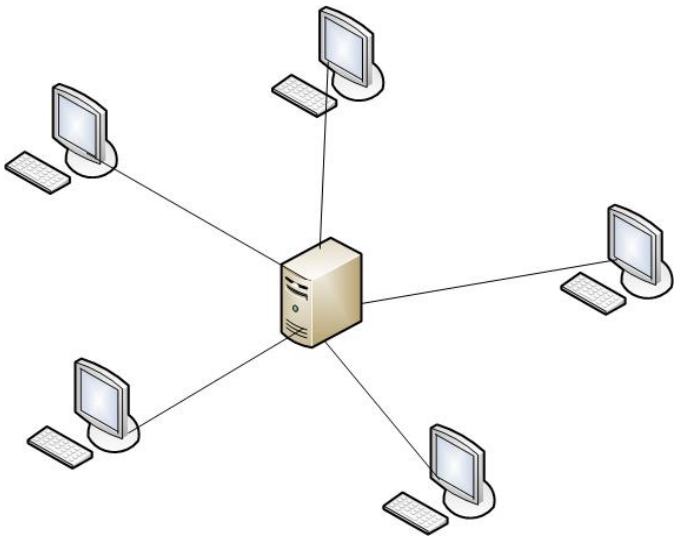


Figura 2.1: Red P2P centralizada

1.3.2 **Descentralizadas**

Son las aplicaciones de segunda generación. No cuentan con un servidor central y todos los nodos tienen el mismo estatus dentro de la red. Las comunicaciones son usuario a usuario con la ayuda de otros que permiten enlazar las comunicaciones.

Su funcionamiento consiste en que, un nodo envía una petición a otro nodo, este reenvía la solicitud a todos los nodos que conoce y así sucesivamente, añadiendo cada nodo información a la búsqueda y se la reenvía al nodo buscador. Una vez finalizada la búsqueda, se establece una conexión entre el nodo buscador y el nodo buscado directamente sin intermediarios. A pesar de que el número de saltos de la red es potencialmente infinito, permanece limitado por el TTL (tiempo de vida máximo) que se va restando en cada salto que realiza la petición, descartándose cuando llega a cero. Se puede ver su esquema en la Figura 2.2.

Este modelo es mucho más robusto y económico, porque no requiere de un servidor central, pero por contra, es más lento y sobrecarga el ancho de banda a la hora de realizar las búsquedas. Un ejemplo de este tipo de redes es la primera versión del protocolo Gnutella [GNU10].

1.3.3 **Híbridas**

Son las llamadas de tercera generación, porque aprovechan las ventajas de las dos soluciones anteriores (resistencia a ataques y eficiencia). Consiguen esto introduciendo la figura de los *superpeers* (supernodos). Su función es la de realizar las operaciones de búsqueda y transferencia, actuando así como nodos activos y agilizando el funcionamiento de la red. Cualquier nodo puede consultar al supernodo la lista de nodos activos, de esta manera cada nodo sólo mantiene un número pequeño de conexiones abiertas (una de ellas al supernodo). Los supernodos se conectan entre sí para mantener actualizados los índices de búsqueda. Se puede ver su esquema en la Figura 2.3.

Con este tercer modelo, se obtiene una red más escalable mediante la reducción de nodos implicados en las búsqueda, así como la disminución de tráfico entre ellos. La velocidad de respuesta a las solicitudes puede ser cercana a la de los modelos centralizados. Algunos ejemplos del uso de este modelo son las aplicaciones Kazaa [KAZ10] y eDonkey [EDO10].

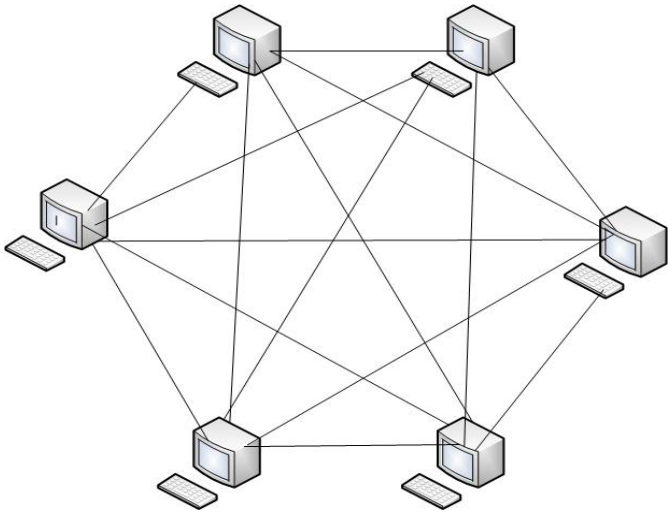


Figura 2.2: Red P2P descentralizada

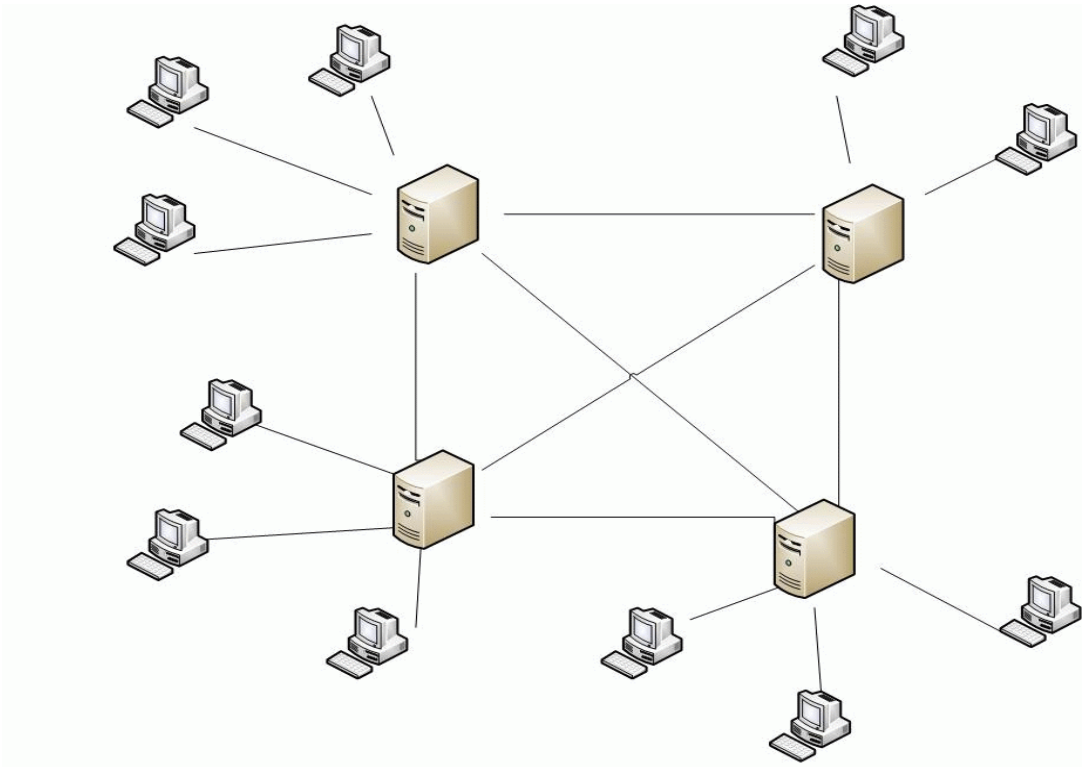


Figura 2.3: Red P2P híbrida

1.4 **Clasificación de redes según su estructuración**

La comunicación entre los peers se realiza a partir de una serie de protocolos. En el caso del nivel de transporte, suele usarse el TCP/IP. Los nodos de una red P2P se organizan en lo que se llama red virtual en capas (overlay network), en la que la topología de conexión a nivel de aplicación no es exactamente igual que la topología física de la red. Este nuevo tipo de red puede ser de dos tipos,no estructurada y estructurada.

1.4.1 **No estructuradas**

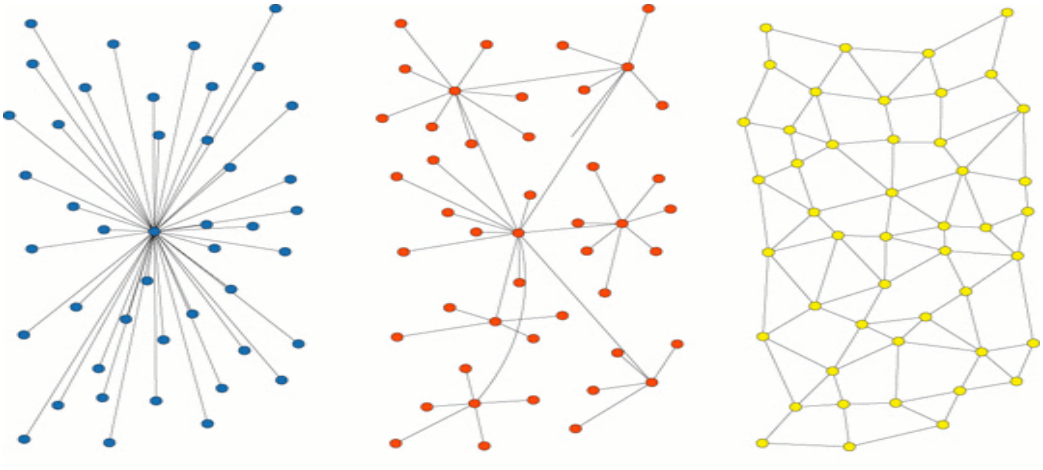
Este tipo de redes son las que se forman arbitrariamente, estableciendo los enlaces entre nodos en base a que se conozcan o no, sin ningún otro tipo de criterio. La redes se pueden construir de manera muy fácil, únicamente un usuario que quiera unirse a la red copiará los enlaces que existan desde un nodo a los demás que este conozca y después formar sus propios enlaces. El inconveniente de estas redes viene a la hora de encontrar una información y el grado de popularidad de esta, ya que, si no es muy popular y debido a la falta de correlación entre usuarios, la petición deberá recorrer muchos nodos llegándose incluso a no resolver en algunos casos. Estas peticiones sobrecargan la red, haciéndola menos eficiente.

1.4.2 **Estructuradas**

Este tipo de red se implementa sobre todo en las redes descentralizadas, para mejorar la infraestructura de búsqueda y almacenamiento estable del que carecen. Para ello, se usan las llamadas DHT (Distributed Hash Tables), que son un tipo de algoritmos y sistemas de búsqueda distribuidos y descentralizados.

En un sistema de varios nodos implementando DHT, la tabla hash es distribuida uniformemente entre todos los nodos, almacenando cada uno parte de ella. Los datos del sistema reciben claves numéricas unívocas generadas con algoritmos hash. El objetivo de DHT es localizar rápidamente los datos utilizando las claves. Los nodos en DHT forman una red virtual en capas con una topología organizada, por eso se les llama redes estructuradas. Un protocolo de afiliación permite a un nuevo nodo integrarse en el sistema ya existente. Este protocolo introduce al nodo en un conjunto de vecinos y facilita la construcción de una tabla de encaminamiento de nodos nueva.

Se puede decir que las redes estructuradas superan las limitaciones de las redes no estructuradas, sobre todo en lo que se refiere a la escalabilidad. Las DHT proporcionan una capa de abstracción eficiente para el encaminamiento y manejo de los datos en sistemas distribuidos.



Red centralizada Red descentralizada Red distribuida

Figura 2.4: Esquemas de los diferentes tipos de redes P2P

1.4.2.1 **Kademlia**

Es un tipo de tabla de hash distribuida que coloca los nodos en forma de árbol binario. Usa la métrica XOR para calcular la distancia entre las diferentes claves, lo que propicia que todas las búsquedas por la misma clave se encaminen hacia el mismo destino. Para cada nodo, se divide el árbol en subárboles que no contienen al nodo. En primer lugar la mitad del árbol, luego la mitad del árbol restante y así sucesivamente. Un ejemplo para un nodo con prefijo 0011 serían los subárboles con las siguientes raíces, 1, 10, 000 y 0010. Kademlia se asegura de que cada nodo conozca al menos a otro nodo en cada uno de los subárboles generados. Gracias a esta filosofía, cada nodo puede preguntar iterativamente a sus nodos conocidos si saben encontrar la clave requerida, obteniendo en cada salto una reducción del árbol mayor o igual a la mitad.

Como vemos en la figura 2.5, siguiendo el ejemplo antes citado, si el nodo con raíz 0011 quiere preguntar por el nodo con raíz 1110, primero deberá preguntar al nodo que conozca del subárbol con raíz 1 (en este caso el 101). Este le guiará al nodo que conozca en el subárbol con raíz 11 (el 1101), el cual a su vez le indicará el siguiente salto dándole información sobre su conocido en el subárbol con raíz 111 (el 11111), el cual ya conoce la dirección del nodo buscado.

Este sistema de búsqueda binario puede ser modificado para cubrir una mayor cantidad de bits en cada salto, de manera que se genere un árbol que no sea binario.

1.5 **Aplicaciones**

Existen muchas aplicaciones sobre distintos tipos de redes P2P, a continuación se comentarán algunas de ellas según la funcionalidad que cumplen.

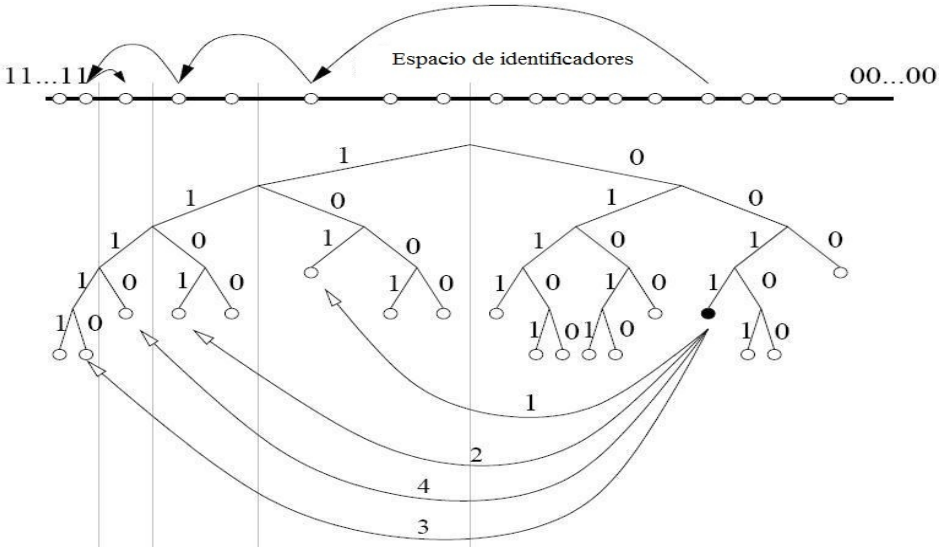


Figura 2.5: Búsqueda del nodo 1110 por el nodo 0011 en Kademlia [1]

- **Intercambio de contenidos.** Son las más famosas y polémicas. Su filosofía está basada en guardar un archivo en algún nodo de la red P2P y ponerlo a disposición de los otros para compartirlo. Los tipos de archivos intercambiados son de muchos tipos, pero especialmente los de música (*mp3*) [MP310] y video (*divx*) [DIV10] son los más intercambiados, lo que ha provocado reacciones contrarias a estas redes debido a los derechos de autor. Sin embargo, se está trabajando con los DRM (Digital Rights Management)[2] para que estos intercambios sean legales y respeten en todo momento el *copyright*.

- **Distribución de contenidos.** La diferencia con el anterior tipo de aplicaciones es que ahora, el contenido se replica en todos los nodos. El video bajo demanda y/o video *streaming* [3] es de este tipo. Cadenas televisivas como la BBC cuentan con este tipo de aplicaciones.

- **Sincronización de datos.** Parecido a distribución de contenidos, pero su función se puede emplear para que fabricantes de software distribuyan sus parches de actualización de forma legal usando redes P2P.

- **Mensajería instantánea, telefonía y videoconferencia IP.** Antes se usaba el modelo cliente-servidor, pero debido al incremento de clientes, coste de servidores y mala calidad en las comunicaciones, se cambió a otro modelo. Un ejemplo de ello es la aplicación privada *Skype* que fue la primera en desarrollar la telefonía IP y de la cual se comentará algún detalle a continuación.

---

[1] Figura obtenida de [KAD10]

[2] La gestión de derechos digitales o DRM es un término que engloba a varias técnicas que permiten al dueño de los derechos o distribuidor de un contenido en formato digital controlar como el material puede ser empleado por los usuarios en cualquier tipo de dispositivo electrónico. Las técnicas DRM se basan en encriptación, la cual permite establecer cómo podrá ser accedido el fichero por los distintos usuarios, incluyendo la cuestión de licencias y desencriptación en el dispositivo del cliente.

[3] En el modo streaming (flujo) no es necesario tener todo el archivo para ser visualizado, si no que puede empezar la reproducción mientras se está descargando lo restante del video.

Skype [SKY10] es una aplicación desarrollada para realizar llamadas a través de Internet usando redes P2P (VoIP). Es una red de telefonía entre pares por Internet que fue creada por los creadores de Kazaa (Janus Friis y Niklas Zennström) en 2003. Su código y protocolo permanecen cerrados y propietarios pero los internautas pueden descargarse gratuitamente la aplicación de la página oficial (http://www.skype.com). Los usuarios pueden hablar entre ellos gratuitamente y también realizar llamadas a teléfonos convencionales, pero en este caso lleva asociado un coste. En este esquema existen los nodos, que son los usuarios y los supernodos que son usuarios con un gran ancho de banda, una IP pública accesible y una CPU con gran capacidad de computación. Su interfaz de usuario es muy parecida a otros programas de mensajería instantánea como Windows Live Messenger [MSN10] o Yahoo! Messenger [YMS10].

1.6 **Redes P2P Jerárquicas**

Las redes P2P son las más conocidas de este tipo. Napster fue el primer programa que usó estas redes y el término peer-to-peer. Una vez desaparecido Napster por motivos legales, aparecieron sistemas como Gnutella o Freenet sustituyendo el servidor central por el método de inundación (preguntar a los ordenadores vecinos si conocían dónde estaba el archivo). El grafo generado a partir de esa búsqueda es un ejemplo de *red overlay*, debido a que dicha red se encuentra a nivel de aplicación con su propia topología y enrutamiento sobre el nivel de transporte de Internet.

Dicho tipo de redes tienen sus ventajas como son la simplicidad, la distribución del control y que evitan tener un único punto de fallo, pero por contra, tienen mucho más tráfico de control y podría ser una solución no escalable.

Se han visto a lo largo de este capítulo los distintos tipos de redes P2P, pero ahora vamos a centrarnos en las características de este tipo de redes así como en sus requisitos. Los nodos se organizar en grupos, y éstos están comunicados a su vez formando otra red overlay. Este tipo de redes son una evolución de las redes estructuradas, ya que permiten una solución más escalable teniendo un número elevado de nodos.

Los grupos formados en estas redes se organizan según la aplicación para la cual se vaya a usar dicha red overlay. Un ejemplo de ordenación puede ser la proximidad geográfica.

En el caso que se está explicando en este documento, se tiene una red de dos niveles. El primer nivel lo componen los superpeers y forman la llamada red de interconexión y el segundo lo forman los nodos normales y se llama dominio o cluster. Un ejemplo se puede ver en la figura 2.6.

Debido a que se tiene una arquitectura con dos niveles, para enrutar los paquetes a través de la red, se usan identificadores únicos tanto para los peers como para los dominios. Cada dominio tiene uno o más superpeers que realizan la función de gateways entre los grupos, por este motivo tienen más carga computacional y de red que los nodos normales.

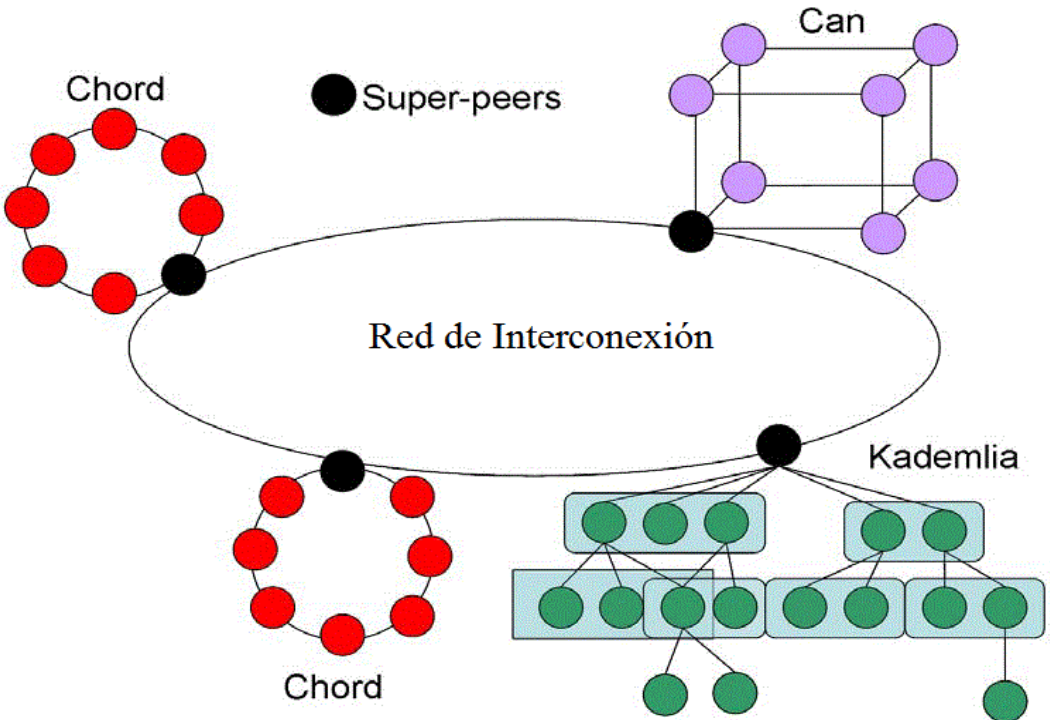


Figura 2.6: Ejemplo de red overlay [4]

Para un buen funcionamiento, es necesario que cada nodo conozca la IP de al menos un superpeer de su dominio para poder comunicarse con los peers de otro dominio. Cumpliéndose esto, cuando un peer quiera realizar una operación de *look-up* (búsqueda) sobre un valor que no se encuentre en su dominio, envía una *query* (consulta) a un superpeer de su dominio que lo encaminará a un superpeer del dominio buscado. Si un nodo quiere buscar un valor en otro dominio, pero no conoce la IP de ningún superpeer, se lo manda a otro nodo normal dentro de su dominio y así sucesivamente hasta que alguno conozca la IP de algún superpeer de su dominio.

1.6.1 **Requisitos de una red overlay jerárquica**

Una red overlay jerárquica tiene una serie de particularidades que la hacen diferente a una red overlay plana. Estas peculiaridades se pueden traducir en la siguiente serie de requisitos:

- **Separación de los peers en grupos.** La red tiene que constar de grupos o clusters de peers, y a su vez, estos grupos tiene que estar conectados entre sí por una red de superpeers.

[4] Figura obtenida de [MYBC+09]

- **Independencia entre grupos.** Cada grupo de la red debe ser independiente, pudiendo tener distintos protocolos de enrutamiento en cada uno de ellos. En este caso, el protocolo utilizado es el P2P estructurado Kademlia [KAD10] para que funcione de forma jerárquica.

- **Uso de identificadores jerárquicos.** Se usan identificadores que permiten diferenciar la información de red, de la del usuario. La primera parte del identificador será el hash del dominio y la segunda el hash asociado a la información guardada (en una aplicación de VoIP será la información del usuario). Posteriormente se mostrará con más detalle la identificación usada en este proyecto.

- **Peers y superpeers participan en el enrutamiento.** Se tendrá pues, que para cada peer y query, si la primera parte del identificador es el del propio peer, el mensaje se enrutará dentro del propio cluster. Si es diferente, el peer lo enrutará a un superpeer de su dominio y este buscará el dominio al que pertenece y se lo enviará al superpeer de dicho dominio. Finalmente el superpeer del dominio al que pertenece buscará en su dominio al usuario buscado.

1.6.2 P2PP

Antes de comenzar a hablar sobre la modificación del protocolo P2PP jerárquico sobre el que están basados los experimentos de este proyecto, anteriormente se va a proceder a comentar algunas consideraciones para facilitar la comprensión del punto a tratar.

1.6.2.1 Implementación utilizada

Entre todas las opciones barajadas, se eligió la implementación P2PP (Peer-to-peer Protocol) [P2PP09] porque se ajustaba mejor a los requisitos necesarios. Proporciona una red P2P, tiene código libre en C++ y está orientado a objetos, lo cual fue muy factible a la hora de realizar las modificaciones pertinentes.

P2PP es un protocolo a nivel de aplicación que puede ser usado para formar y mantener una overlay entre nodos participantes. Es un proyecto desarrollado por la Universidad de Columbia, que provee mecanismos para que los nodos puedan unirse, abandonar, publicar y buscar recursos en la overlay.

La overlay puede ser creada usando protocolos P2P estructurados o no estructurados como pueden ser Bamboo, Chord, Pastry, Kademlia, Gnutella y Gia. P2PP usa transporte seguro, tiene una API para aplicaciones, mecanismos para atravesar NATs y firewalls, intercambio de capacidades de los nodos y de información de diagnóstico. Al igual que P2PSIP, P2PP está diseñado para soportar una red SIP P2P, pero puede ser usado para otras aplicaciones. Existen dos versiones. Está orientado a objetos y permite fácilmente la introducción de nuevos protocolos. Este proyecto esta muy bien documentado y ha sido probado en el entorno PlanetLab con aproximadamente 1000 nodos.

1.6.2.2 **Estructura de la implementación utilizada**

Aunque existen dos versiones de P2PP, se decidió utilizar la versión 0.1 porque aunque la 0.2 es capaz de atravesar NATs (usando la librería PJNATH), desde la web se indica que es mejor usar la 0.1 si esta funcionalidad no se va a utilizar.

1.6.2.3 **Modificaciones realizadas**

En lo que refiere al protocolo, se tuvieron que modificar algunas clases y crear otras nuevas. La estructura original tenía una clase Peer que heredaba de la clase Node. Se añadió una clase intermedia llamada Hpeer entre la clase Peer y la clase que implementa el comportamiento propio de cada protocolo. Esta clase añadió dos funcionalidades básicas a la clase Peer.

La primera funcionalidad era la de crear ID jerárquicos. Se necesitaba que el routing dentro de nuestro cluster se hiciera en función del *suffix ID* y que en la red de superpeers se hiciera mediante el *prefix ID*. Por tanto, los peers que se encuentren en el mismo dominio tendrán el mismo prefix ID y dependiendo si es un peer o un recurso distinto suffix ID.

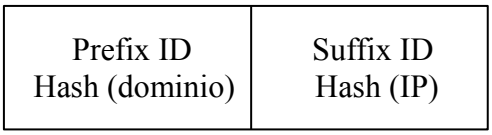


Figura 2.7: Identificador jerárquico de los peers

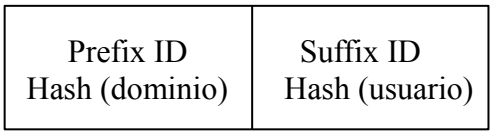


Figura 2.8: Identificador jerárquico de los recursos

En cuanto a los superpeers, se han usado identificadores normales, ya que en este caso, el dominio es el recurso a buscar. Por ello, el ID de los superpeers es el hash de su IP y el ID de los recursos el hash del dominio.

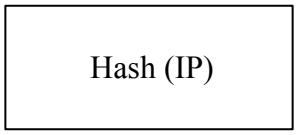


Figura 2.9: Identificador de los superpeers

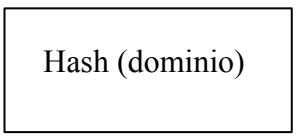


Figura 2.10: Identificador de los recursos en la red de interconexión

La segunda funcionalidad introducida es la de detectar cuando una petición es para el propio dominio y cuando se debe reenviar a la red de superpeers, eso se consiguió modificando diferentes métodos que no se van a entrar a explicar debido a que se encuentran detalladamente comentados en [SENB07a].

2 **ModelNet**

En este segundo punto se va a explicar por qué se ha utilizado y qué es ModelNet [MOD10]. Se ha usado principalmente porque es el emulador utilizado para realizar los experimentos de la aplicación de partida y también porque es un entorno muy fácil y sencillo en el que se tienen todas las máquinas virtuales necesarias para ejecutar los experimentos. Sin él, la administración de todas las máquinas físicas como Hera, Carcoma, etc, así como las máquinas virtuales que en su interior se encuentran como core1, edge5, ect, hubiera sido un trabajo muy costoso y poco eficiente en lo que al tiempo se refiere. En cambio, con ModelNet se tiene todo esto en el mismo interfaz y de esta forma la posibilidad de realizar las tareas de administración más eficientemente por parte del desarrollador.

De ModelNet se puede decir que es un emulador de red a gran escala, que permite a los usuarios evaluar sistemas distribuidos de red en entornos realistas como Internet. También permite probar prototipos sobre sistemas operativos sin modificar, a través de distintos escenarios de red. Es también capaz de combinar la repetitividad de una simulación con la fiabilidad de un entorno real. La comunidad de usuarios de ModelNet ha desarrollado y probado la distribución de contenidos en la red, sistemas P2P, protocolos de capa de transporte, etc.

Básicamente, el funcionamiento es: los usuarios despliegan ModelNet en su cluster local. Cada instancia de la aplicación se ejecuta en un nodo virtual. ModelNet distribuye los nodos virtuales a través de un conjunto de equipos físicos llamados *edge nodes*.

El sistema configura los edge nodes para encaminar sus paquetees a través del núcleo ModelNet (una o más máquinas físicas). Soporta emulación hop-by-hop, captura los efectos del tráfico cruzado y la congestión dentro de la red.

Aún teniendo todas esas características, usar ModelNet es muy sencillo y requiere los siguientes pasos:

- **Especificación de topología.** La topología puede ser de cualquier tipo. Puede estar desde hecha a mano a crearse por un generador de topologías como GT-ITM, Brite, INET o Orbis.

- **Formación.** El sistema de mapas de nodos virtuales da a cada nodo virtual una dirección IP apropiada en la topología (en el rango 10.1.X.X). Teniendo una CPU relativamente lenta, es posible hacer correr entre 10 y 100 instancias en un edge node. ModelNet construirá el camino más corto en la “tabla de rutas” que contendrá el conjunto de rutas a atravesar por cada par origen-destino.

- **Implementación.** Creando scripts por ejemplo con gexec, permite al usuario desplegar tanto la topología como el código de los cores y los edges nodes. ModelNet es capaz de realizar experimentos con más de 100 máquinas virtuales con tan sólo dos equipos físicos.

Para ver detalladamente como se instala ModelNet en las máquinas, ver el Apéndice B.

3 XML

Hasta este punto, se han explicado los dos puntos más importantes para la comprensión de la aplicación desde la que se parte para poder realizar este proyecto. Por un lado están las redes P2P, que son las que se van a usar para lanzar los experimentos, y por otro lado ModelNet, que es el emulador utilizado para poder monitorizar y administrar todas las máquinas que forman el escenario de pruebas de una manera eficiente.

A continuación, los dos siguientes puntos a priori no tienen mucha relación con los anteriores, pero para este proyecto son los más importantes, y justamente son la muestra de la gran diferencia existente entre los objetivos buscados en ambos casos. En el primer proyecto se buscó validar una aplicación de partida, y en este, el objetivo buscado es, como se ha dicho en repetidas ocasiones, el de generalizar dicha aplicación de partida para que cualquier desarrollador pueda usar todo su potencial con un grado mucho menor en lo que a la complejidad se refiere. Este gran recorte en la complejidad se ha logrado usando dos lenguajes de programación muy conocidos como son XML y Java, pero sobre todo en la capacidad que tienen ambos para trabajar conjuntamente, convirtiendo por ejemplo, un archivo XML en parámetros que regirán una ejecución Java.

En primer lugar se va a explicar qué es XML y las características más importantes en relación con el uso que se le ha dado en este proyecto. En el siguiente punto se hará lo propio con Java.

3.1 Introducción

XML (Extensible Markup Language) es un metalenguaje de etiquetas desarrollado por el W3C (World Wide Web Consortium) [W3C10], lo que significa que no sólo es un lenguaje, si no que también es una forma de especificar lenguajes de marcación. Es un subconjunto de SGML (Structured Generalized Markup Language) [SGM10]. Este, es un estándar ISO para documentos estructurados sumamente complejo para poder servir documentos en la web.

La W3C define unos objetivos de diseño de XML, los cuales son:

1. Debe ser directamente utilizable por Internet.
2. Debe soportar una amplia variedad de aplicaciones.
3. Debe ser compatible con SGML.
4. Debe ser fácil la escritura de programas que procesen documentos XML.
5. El número de características opcionales en XML debe ser mínima, idealmente cero.

6. Los documentos XML deben ser legibles por humanos y razonablemente claros.
7. El diseño XML debe ser preparado rápidamente.
8. El diseño XML debe ser formal y conciso.
9. Los documentos XML deben ser fácilmente creables.
10. La semántica en las marcas XML es de mínima importancia.

XML cambia el paradigma de programación: antes era basada en funciones u objetos y ahora la programación es basada en el documento.

Dentro del estándar XML se definen tres tipos de documentos:

INVÁLIDOS: No siguen las reglas del estándar en cuanto a las etiquetas o bien, disponen de DTD y no se atiene a las reglas indicadas en el mismo.
BIEN FORMADOS: Cumplen las reglas de estándar pero no tienen DTD asociado.
VÁLIDOS: Cumplen tanto las reglas del estándar como las del DTD asociado.

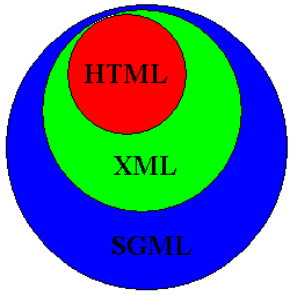


Figura 2.11: Relación entre SGML, XML y HTML

Las principales ventajas de este lenguaje son:

- Independencia del sistema operativo y aplicación que lo usarán.
- Es un metalenguaje, no un lenguaje de marcado específico.
- Permite indicar lo que se está representando en sus documentos mediante las etiquetas.
- Separa datos y presentación, permitiendo tratar documentos más grandes y complejos.
- Puede utilizarse en muchos ámbitos, no sólo en Internet y dentro de un navegador web.

3.2 **DTD**

El DTD (Document Type Definition) es una parte importante dentro de este lenguaje porque permite validar documentos XML. En él se define exactamente la gramática del documento XML; es decir, tipos de elementos, atributos y entidades permitidas. También puede expresar algunas limitaciones para combinarlas.

La dinámica a seguir cuando se procesa cualquier información formateada mediante XML es, en primer lugar, comprobar si está bien formada, después, ver si sigue sus reglas gramaticales y por último si incluye o no referencia a un DTD.

Los DTDs tienen ciertos inconvenientes. Uno de ellos es que están basados en una sintaxis especializada, otro es que soportan únicamente una forma primitiva de agrupación a través de entidades de parámetros, de ahí, que existan también los XML Schemas. Estos superan todas estas limitaciones, describen la estructura del documento XML, siendo un mecanismo similar a los DTDs pero dotando a los XML de más potencia y expresividad en sus estructuras. También cabe destacar que se basan en XML con todas las ventajas que esto conlleva y que soportan mayor cantidad de tipos de datos.

3.3 **Parsers para procesar XML**

Hay que decir que los documentos XML por si solos no tienen valor, por eso es necesario procesarlos para extraer la información que contienen, para ello se usan los parsers. Estos son procesadores o analizadores sintácticos que tratan un documento XML y verifican que está bien formado; además algunos parsers también comprueban que el código XML sea válido.

Estos elementos son la herramienta principal de cualquier aplicación XML. Se pueden incorporar a nuestras aplicaciones, de manera que estas manipulen, trabajen y creen documentos XML.

Existen diferentes tecnologías para poder procesar documentos XML, tales como:

- SAX, Simple API for XML.
- DOM, Document Object Model API from W3C.
- XLST, XML Style Sheet Language Transformations from W3C.
- Xpath, XML Path Language from W3C.
- JDOM, “Java Optimized” Document Object Model API from jdom.org

3.4 **Modelos para procesamiento XML**

Son las fases con las que se pueden procesar documentos XML, recuperar información y generar documentos XML resultantes. Estas son:

- Procesamiento de entrada XML

- Analizar y validar el documento fuente.
- Reconocer/buscar información importante basándose en su localización o en su etiquetado en el documento fuente.
- Extraer la información importante una vez que se ha localizado.

➢ Procesamiento de salida XML

- Construir un modelo de documento para generarlo con DOM, JDOM, SAX, etc.
- Aplicar hojas de estilo XSLT o directamente generar un documento de salida XML.

Para poder entender mejor los parsers o modelos de procesamiento de entrada XML se explicarán detalladamente los más comunes, SAX y DOM. Después se hará una comparativa entre ambos y posteriormente, se explicará XSLT, que es un modelo de procesamiento XML de nivel superior, en donde el desarrollador codifica reglas (plantillas) que se aplicarán a los XML.

3.4.1 **SAX [SAX10] (Simple API for XML)**

Esta API (Application Programming Interface) [API10] usa un modelo basado en eventos, lo cual permite el procesamiento de un documento fuente como un flujo de eventos. Los eventos son disparados mientras se analiza como un flujo continuo de llamadas e invocaciones a métodos, lo que obliga a hacer todos los pasos (nombrado, análisis, reconocimiento, extracción y mapeo) en un solo ciclo. Los elementos están anidados de la misma forma que los elementos en el documento; por tanto, no se crea ningún modelo de documento intermedio. Esto último tiene sus ventajas e inconvenientes, por un lado el uso de memoria es bajo, pero en cambio el modelo de programación necesario ha de ser más complejo. Esta filosofía tiene la debilidad de que no puede utilizarse cuando un modelo de documento tiene que ser editado o procesado varias veces.

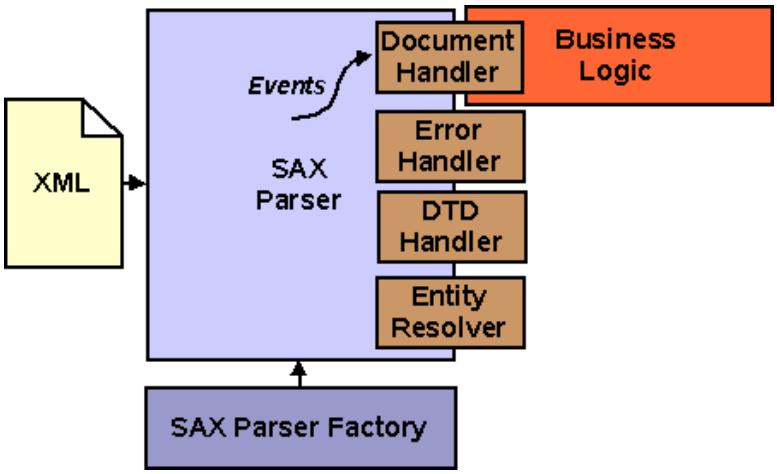


Figura 2.12: Estructura SAX

3.4.2 **DOM [DOM10] (Document Object Model)**

Esta tecnología es una especificación W3C, la cual dice de ella que "es una interfaz neutral del lenguaje y la plataforma, que permitirá a los programas y *scripts* acceder dinámicamente y actualizar el documento, su estructura y su estilo". Con ella, lo que hay que hacer es, escribir un código para pasar a través de una estructura de datos tipo árbol (creada por el analizador) desde el documento fuente. Al contrario que en SAX, el analizador DOM hace al menos dos ciclos. En primer lugar, crea una estructura de datos tipo árbol que modela el documento fuente XML (árbol DOM), y posteriormente para buscar la información relevante se pasa a través de dicho árbol y se procesa. Este último ciclo se puede repetir tantas veces como sea necesario mientras el árbol DOM exista en memoria. Al igual que SAX, DOM también tiene sus ventajas e inconvenientes. Al tener una estructura intermedia de árbol en memoria, el uso de esta es mayor que en para SAX. A favor tiene que una vez construido el árbol DOM, se puede acceder y procesar múltiples veces.

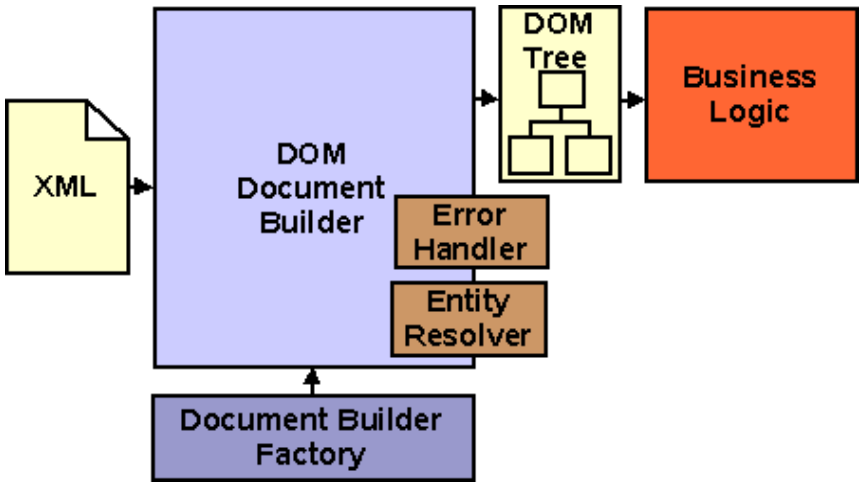


Figura 2.13: Estructura DOM

3.4.3 **CARACTERÍSTICAS SAX Y DOM**

SAX	DOM
Modelo basado en eventos	Estructura de datos tipo árbol
Acceso serie (flujo de eventos)	Acceso aleatorio (estructura de datos en memoria)
Bajo uso de memoria (sólo genera eventos)	Alto uso de memoria (todo el documento en memoria)
Sólo procesa partes del documento	Para editar el documento
Sólo una vez para procesar el documento	Múltiples veces para procesar el documento

Tabla 2.1: Comparación entre SAX y DOM

3.4.4 **XSLT (XML Style Sheet Language Transformations from W3C)**

Como se comentó anteriormente, XSLT [XSL10], es un modelo de procesamiento XML de nivel superior a los anteriores, debido a que el programador codifica unas reglas que se aplicarán a los XML. Al indicar que es de nivel superior, se está diciendo que es un lenguaje para describir cómo transformar un documento XML en otro. Permite definir las reglas que serán aplicadas cuando se encuentren los patrones especificados en el documento fuente e insertar elementos en el árbol resultante.

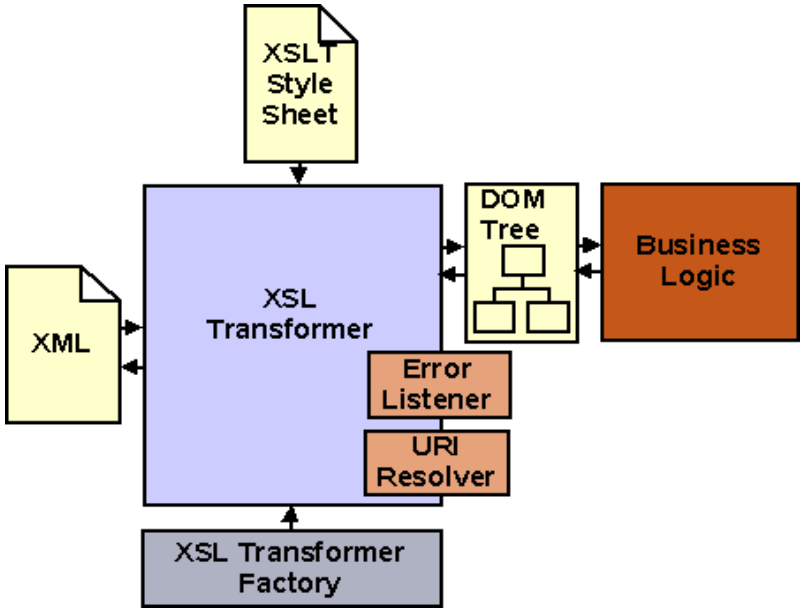


Figura 2.14: Estructura XSLT

Para tener una visión más general de las tecnologías que se han explicado anteriormente, en la figura 2.15 se pueden ver las APIs Java disponibles para procesar XML. Se pueden construir unas encima de otras para añadir más abstracción y por tanto más potencia a la aplicación desarrollada. Las flechas indican las dependencias entre ellas.

Una vez explicados los modelos de procesamiento de entrada XML, y aunque en el proyecto que se está explicando en este documento dicha funcionalidad no se va a usar, debido a que lo que se hace es partir de un documento XML con múltiples datos, procesarlo y utilizar dichos datos para configurar nuestra aplicación, no está demás indicar brevemente los modelos de procesamiento de salida XML. Existen dos soluciones:

- Generar el documento XML directamente “a mano”.
- Construir un árbol DOM ó JDOM correspondiente.

De estas soluciones, la segunda sería la más preferible en cuanto a la formación del documento y un procesamiento posterior más eficiente, pero peor en cuanto a que requiere más recursos de memoria.

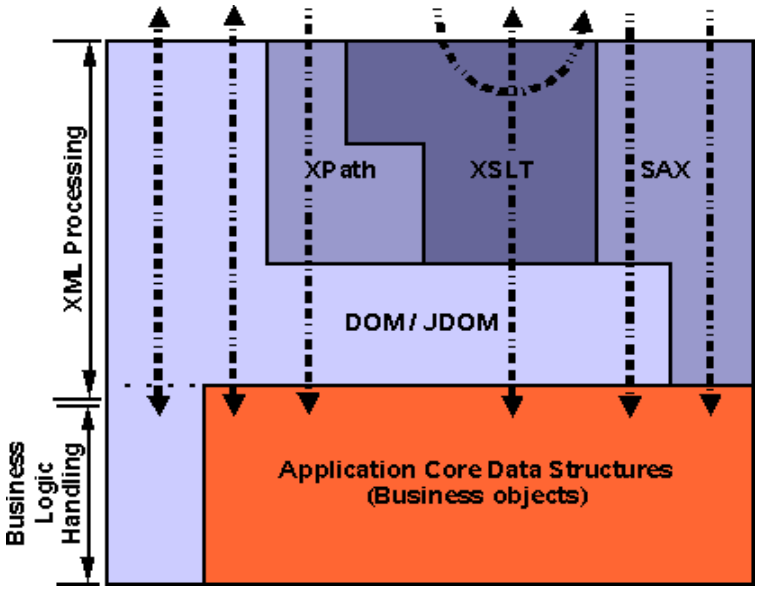


Figura 2.15: Relación entre tecnologías para procesar XML

3.5 Modelos de parsers (JDOM)

Existen muchos modelos de parsers tales como JAXP [JAX10] (Java API for XML Processing), CRIMSON [CRI10], XERCES [XER10], XMLDOM [XMD10], Parser XP [PXP10] y JDOM [JDOM10] entre otros. En este documento y debido a la elección que se ha hecho para el proyecto, se explicará el parser JDOM y sus características.

A pesar de existir diversos parsers para aplicaciones Java, tecnologías como SAX y DOM no fueron diseñadas con Java en mente, de ahí que JDOM aprovechara ese vacío y creara una metodología para procesar XML más acorde con el mundo Java.

Se ha elegido JDOM para la realización de este proyecto por varios motivos, uno de ellos es que es una API pensada específicamente para el procesado de documentos XML con Java y esto no sucede con SAX y DOM. Tiene varias similitudes con DOM, como por ejemplo que también se basa en el parseado de un documento XML y la construcción de un árbol de elementos, atributos, comentarios, instrucciones de procesamiento, etc. Una vez construido el árbol, se puede acceder directamente a cualquiera de sus componentes. Se ha elegido JDOM en vez de DOM porque al estar pensada expresamente para Java, conlleva que elementos, atributos, etc, se representan en Java mediante clases sin que exista el concepto de Nodo como en DOM, siendo este uno de los motivos de no elegir DOM. Tampoco se eligió SAX porque tal y como se vio en el apartado 3.4.1 la forma de procesar la información por eventos no era útil para este caso porque se tenía la necesidad de acceder a la información tantas veces como fuera necesario para poder configurar todas las pruebas correctamente.

Por tanto, JDOM permite al programador el parseado, creación, manipulación y serialización de documentos XML. No incluye un parseador pero puede utilizar uno existente (como XERCES que se basa en SAX) para construir el árbol de elementos JDOM. También cabe mencionar que existen mecanismos para realizar funciones como convertir DOM a JDOM o viceversa, crear un árbol JDOM a partir de SAX o volcar un árbol JDOM hacia SAX.

Para ver mejor las posibilidades de esta tecnología y teniendo en cuenta que ha sido la elegida a la hora de desarrollar el proyecto, se mostrarán las ventajas e inconvenientes que posee:

- Ventajas:

- Diseñado para Java.
- Facilidad de uso (para programadores Java).
- Representación de documentos como árboles, por tanto, nos permite el acceso aleatorio a cualquier parte del documento (ventaja sobre SAX).
- API de libre distribución.

- Desventajas:

- Necesidad de procesar y almacenar en memoria todo el documento; por tanto, mayor consumo de memoria que SAX.
- Debido a su diseño basado en Java, escasa portabilidad a otros lenguajes de programación.
- Al carecer del concepto de NODO, dificulta en cierta medida la navegación a través del árbol del documento. Por ejemplo, al obtener los hijos de un elemento (como resultado es una lista de objetos genéricos java.util.List), debemos comprobar que representan esos objetos (haciendo por ejemplo un instanceOf) para saber si son elementos, atributos, etc.

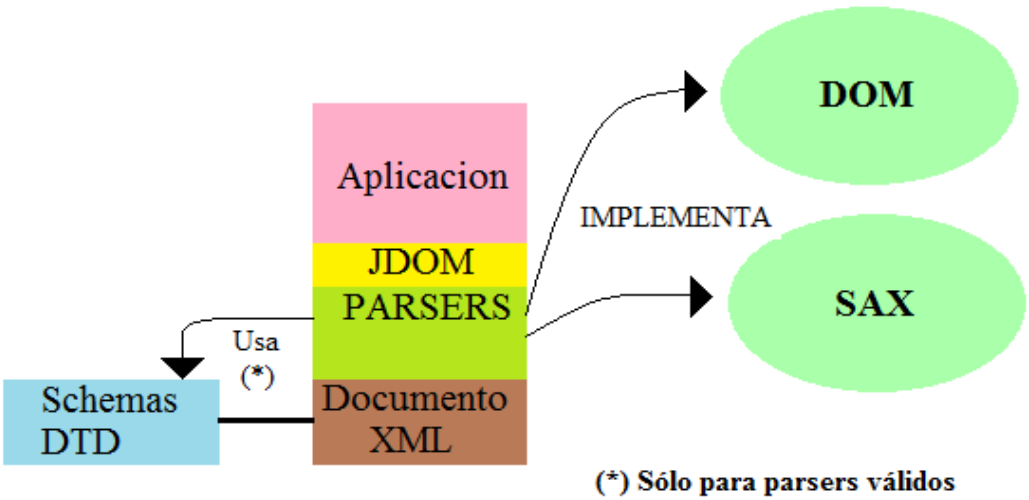


Figura 2.16: Metodología de trabajo con JDOM

4 JAVA 

Como ya se comentó al inicio del apartado 3, XML y Java en una primera impresión pueden parecer no muy acordes con la temática del proyecto, pero en cambio, son fundamentales en el objetivo buscado de generalizar la aplicación de partida. Cabe destacar que todas las clases de la aplicación de partida están desarrolladas en este lenguaje y que por tanto, las modificaciones y creación de nuevas funciones en este proyecto también se han realizado en dicho lenguaje. Pero sobre todo, la mejor opción que ofrecen estos dos lenguajes de programación, ha sido la posibilidad de combinarlos y usarlos conjuntamente para extraer la información contenida en nuestro XML principal del que luego se hablará, para poder usarla como parámetros en nuestras clases Java y así configurar las máquinas, instalar todo lo necesario en ellas y ejecutar los experimentos.

4.1 Introducción

Java [JAV10] es un lenguaje de programación desarrollado por SUN MICROSYSTEMS [SUN10] con la intención de competir con Microsoft [MIC10] en el mercado de la red. Sin embargo, su historia se remonta a la creación de una filial de SUN (First Person) enfocada al desarrollo de aplicaciones para electrodomésticos. Esta filial desapareció tras un par de éxitos de laboratorio y ningún desarrollo comercial.

Uno de los trabajadores de First Person, James Gasling [JAM10], desarrolló un lenguaje derivado de C++ que intentaba eliminar las deficiencias del mismo, lo llamó OaK. Cuando SUN abandonó el proyecto de First Person, se encontró con este lenguaje y tras varias modificaciones (entre ellas el nombre) decidió lanzarlo al mercado el verano de 1995.

El éxito de Java reside en varias características. Es un lenguaje sencillo (dentro de la complejidad de la programación orientada a objetos), es independiente de la plataforma en la que se use y destaca su seguridad (basada en la integridad del código intermedio que se obtiene al compilar el archivo fuente Java). También destacan, su capacidad multihilo y lo integrado que tiene el protocolo TCP/IP, lo que lo hace un lenguaje ideal para Internet.

4.1.1 Orientación a objetos

Los primeros lenguajes de programación exigían pensar en términos de ordenador y no en términos del problema a solucionar. Esto provocaba que los programas fueran difíciles de crear y de mantener, al no tener una relación obvia con el problema que representaban. Muchos paradigmas de programación intentaron resolver este problema alterando la visión del mundo y adaptándola al lenguaje, pero estas aproximaciones moldeaban el mundo como un conjunto de objetos o de listas. Funcionaban bien para unos problemas pero no para otros. Los lenguajes orientados a objetos más generales, permiten realizar soluciones que, simplemente leídas, describen el problema y además permiten escribir soluciones pensando en el problema y no en el ordenador que debe solucionarlo en último extremo. Dichos lenguajes se basan en cinco características:

- **Todo es un objeto.** Cada elemento del problema debe ser modelizado como un objeto.
- **Un programa es un conjunto de objetos diciéndose entre sí qué deben hacer por medio de mensajes.** Cuando se necesita que un objeto haga algo, se le manda un mensaje (ejecutando un método de dicho objeto).
- **Cada objeto tiene su propia memoria, que llena con otros objetos.** De este modo se incrementa la complejidad del programa, pero detrás sólo hay simples objetos.
- **Todo objeto tiene un tipo.** Cada objeto, es una instancia (en caso particular) de una clase (el tipo general). Lo que diferencia a una clase de otra es la respuesta a la pregunta, ¿qué mensaje puedes escribir?
- **Todos los objetos de un determinado tipo pueden recibir los mismos mensajes.** Por ejemplo, un objeto de tipo Cuadrado es también un objeto de tipo Figura, se puede hacer código pensando en los mensajes que se mandan a una figura y aplicarlo a todos los objetos de este tipo, sin pensar si son cuadrados o no.

4.1.2 **Portabilidad**

Cuando se enumeraron las características de Java, se hacía mención a la portabilidad. Esta valiosa característica tiene sus pros y sus contras.

La forma en que Java la consigue es por medio de la compilación, es decir, compilando el lenguaje de programación se pasa a otro lenguaje intermedio que es más cercano al lenguaje máquina (independiente del ordenador y del sistema operativo en que se ejecuta) llamado bytecodes [BYT10] y, finalmente, se interpreta dicho lenguaje por medio de un programa denominado máquina virtual JAVA (JVM).

Una vez dicho esto, hay que entrar a matizar varios puntos para poder comprender mejor los problemas de esta. Se puede poner como ejemplo, que es muy complicado con una única tecnología abarcar todas las gamas de dispositivos que existen (desde un complejo servidor de red hasta un sencillo horno o frigorífico). Por eso, SUN MICROSYSTEMS en 1999 decidió dividir la tecnología Java en varias ediciones. Partiendo de esta decisión, SUN definió el lenguaje Java como Plataforma Java2 y redistribuyó lo que tenía en tres ramas, cada una orientada al sector correspondiente. Cada rama posee su conjunto de APIs y herramientas de desarrollo propias, pero manteniendo el lenguaje y otros aspectos en común:

1 - **Java 2 Standard Edition (J2SE)**. Orientada a los ordenadores de sobremesa. Comprende el JDK (Java Development Kit) y un conjunto de clases para facilitar el desarrollo de aplicaciones, donde la interfaz de usuario tiene gran importancia.

2 - **Java 2 Enterprise Edition (J2EE)**. Engloba a J2SE y lo potencia añadiéndole clases para el desarrollo en entornos corporativos. Esta edición está orientada al desarrollo de componentes y distribución de aplicaciones.

3 - **Java 2 MicroEdition (J2ME)**. Subconjunto de J2SE. Está principalmente orientado para el desarrollo de aplicaciones Java destinadas a dispositivos con recursos limitados como teléfonos móviles y PDAs.

Capítulo III

DESARROLLO Y DISEÑO DE LA APLICACIÓN

1 Introducción

En este capítulo, y una vez que se han comentado en el capítulo anterior todas las tecnologías y herramientas necesarias, se va a explicar detalladamente en que consiste y en base a que criterios se ha diseñado el proyecto que en este documento se explica.

En primer lugar, se va a mostrar y explicar todo el equipo necesario, tanto el hardware como el software con el que se ha contado para llevar a buen puerto el diseño y validación de la herramienta desarrollada. En cuanto a los ordenadores, se va a mostrar la diferencia entre los equipos físicos y las máquinas virtuales creadas dentro de estos para dotar al escenario de una mayor complejidad y a la vez aprovechar mejor todo su potencial.

A continuación, sabiendo con el equipo que se cuenta, se va a entrar en más detalle a explicar la aplicación desde la que se parte, para poder identificar más fácilmente los posibles puntos de mejora que tiene según nuestro criterio.

El objetivo primordial de este proyecto es mejorar una aplicación ya creada haciendo que esta sea más fácil de usar y por tanto más atractiva de utilizar para desarrolladores e investigadores. Por tanto, después de explicar la aplicación de partida, se van a comentar los puntos que se han mejorado, cambiado o simplemente retocado debido a que en el momento del diseño inicial no se tenía como objetivo que la aplicación fuera usada para más aplicaciones que la de partida.

En la última parte del capítulo se va a explicar lo más importante de esta aplicación; es decir, el archivo XML que contendrá toda la información necesaria para una correcta configuración de los experimentos. El motivo de explicarlo al final, pese a que es muy importante, no es otro que el de tener una buena visión del funcionamiento de la aplicación con la lectura de las primeras secciones del capítulo para poder entender así el XML más fácilmente.

Para finalizar, se mostrará el archivo DTD que rige el XML central de la aplicación y después la metodología seguida para ver que la herramienta creada se está ejecutando correctamente con los datos particulares de cada experimento.

2 Escenario de pruebas

Como ya se ha comentado anteriormente, este proyecto tiene como fin el generalizar una aplicación que ya se ha probado y que realiza una función determinada. A continuación se va a describir tanto el escenario físico como lógico montado para realizar los diferentes experimentos.

2.1 Esquema físico del sistema

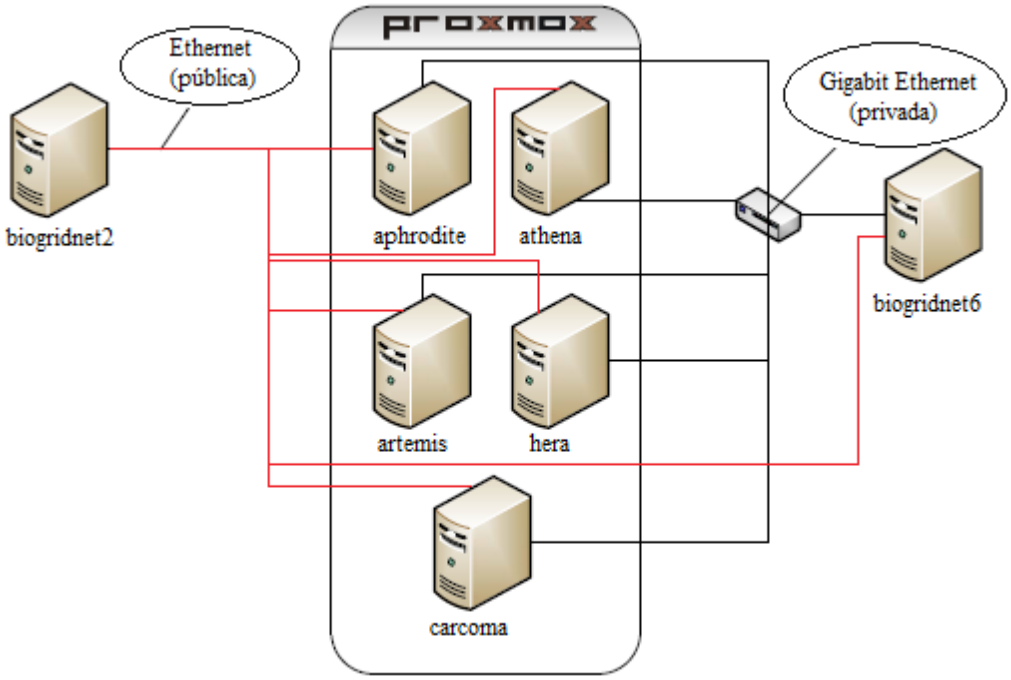


Figura 3.1: Esquema físico del sistema

Como se muestra en la figura, el escenario de pruebas consta de siete máquinas físicas, de las cuales, seis son para emular una red real con ModelNet y la restante (*Biogridnet2*) para desarrollar el software necesario.

Las máquinas que se encargan de emular la red están conectadas entre sí mediante un *switch Gigabit Ethernet* formando una red privada, y además, también se han conectado a la red pública del laboratorio para poder comunicarnse con ellas desde el exterior. La conexión pública entre *Biogridnet2* y *Aprhodite* es la que se usa para configurar e iniciar las pruebas.

Hay que destacar que en cinco de las máquinas utilizadas se ha instalado el emulador de red *Proxmox* [Apéndice C], con el fin de crear en ellas varias máquinas virtuales que ayuden a simular aproximadamente mil nodos virtuales, y poder de este modo, aprovechar todo el potencial de nuestros equipos físicos. Finalmente, se obtiene un sistema lógico diferente al sistema físico que se pasa a mostrar y explicar.

2.2 Esquema lógico del sistema

De este sistema lógico hay que decir que existen tres tipos distintos de máquinas: edge nodes, core nodes y las máquinas de apoyo.

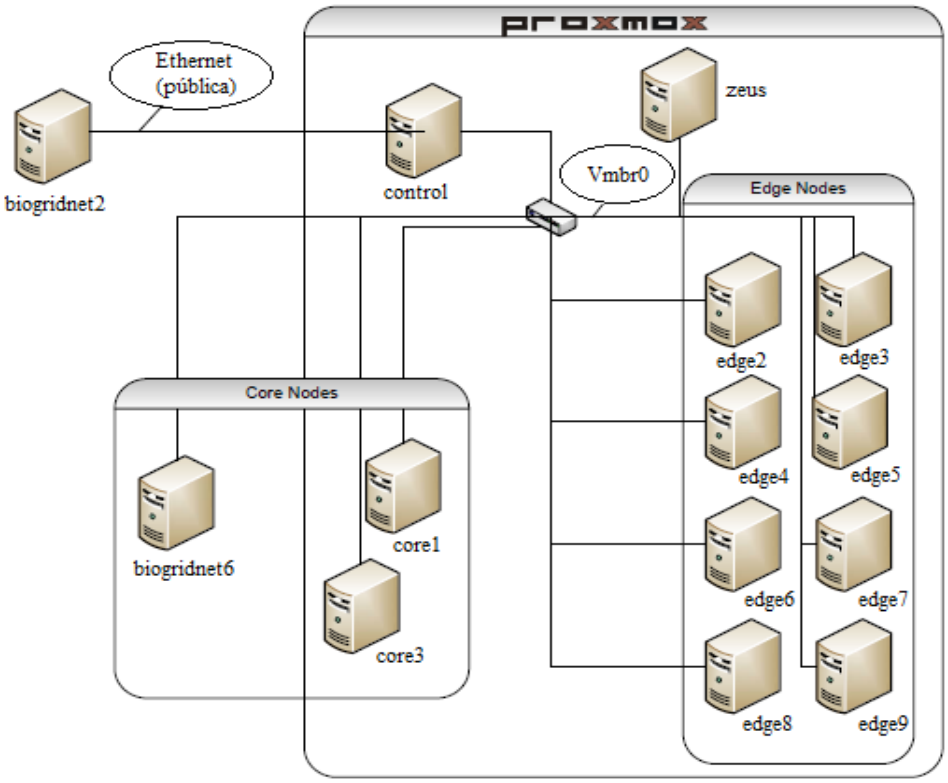


Figura 3.2: Esquema lógico del sistema

En el primer grupo hay ocho máquinas virtuales conectadas entre sí mediante el *bridge* virtual creado por Proxmox (vmbr0).

En el segundo, hay dos máquinas virtuales conectadas al bridge virtual y una máquina física conectada a la red privada.

En el último grupo tenemos tres máquinas. *Zeus* es una máquina virtual que actúa como servidor *DHCP* y *DNS*. *Control* es una máquina virtual muy importante dentro del esquema y que hace de pasarela entre el exterior y la red privada. Desde ella se ejecutan y gestionan las diferentes pruebas de cada experimento. *Biogridnet2* es la máquina física usada para desarrollar el software necesario, así como para ejecutar ciertas funciones durante el proceso de instalación.

Las máquinas virtuales que actúan como edge node están basadas en un sistema operativo Debian 5.0, ejecutándose en un sistema de tipo OpenVZ, en las que se ha instalado la aplicación ModelNet. También se ha instalado Java para facilitar algunas tareas.

En las máquinas que actúan como core node se ha recompilado un FreeBSD 4.8 para ejecutar sobre él los emuladores de ModelNet, pero hay de dos tipos, las máquinas virtuales son del tipo KVM/Qemu mientras que *Biogridnet6* es una máquina real. En ninguna de las tres se ha instalado otra aplicación, ya que interesa que sean lo más simple posible para poder realizar su labor sin problemas.

Por último, las máquinas de apoyo son de diferente naturaleza. *Zeus*, es una máquina virtual de tipo OpenVZ donde se ha configurado un Debian 4.0 para que actúe como servidor DHCP y DNS. *Control*, es una máquina virtual de tipo KVM/Qemu que tiene como sistema operativo un Ubuntu 8.0. *Biogridnet2* es una máquina real con Debian 5.0 sobre la que se ha instalado el software NetBeans [NET10] para ayudar a desarrollar el software y es desde donde se realizan algunas funciones necesarias para completar los experimentos como por ejemplo recopilar toda la información referente a los experimentos que se almacena en la máquina *Control*.

3 Aplicación de partida

Al igual que con el escenario, para la realización de este proyecto se ha partido de una aplicación creada anteriormente [SENB07a] y que para la finalidad buscada en este caso tenía algunos puntos que no la hacían lo suficientemente general. En este apartado se va a a proceder a explicar la aplicación de partida y su funcionamiento, así como los inconvenientes que a nuestro juicio tiene y que han sido el motivo de la realización de este proyecto. En el apartado siguiente se explicarán los puntos de mejora y las soluciones aplicadas, siempre acorde al objetivo principal de nuestro proyecto que no es otro que el de realizar un software capaz de transformar la aplicación de partida particular en una que realice más funciones con un número mayor de posibles variantes.

La aplicación de partida se basa en una serie de aplicaciones en Java que automatizan las tareas de generar, instalar y finalmente ejecutar experimentos. Las principales aplicaciones son tres: Analizador del XML (para generar el gráfico de red que se usará en el experimento), generador de experimentos (crea los archivos necesarios para ejecutar los experimentos) y analizador de resultados (a partir de las trazas generadas por los experimentos permite evaluar los resultados).

Finalmente, los resultados obtenidos se pueden importar a *Matlab* [MAT10] o cualquier otra aplicación para la realización de las pertinentes gráficas y conclusiones.

3.1 Organigrama de funcionamiento de la aplicación de partida

Para poder entender mejor el objetivo de este proyecto y sobre todo, las modificaciones, tanto de criterio como de código realizadas respecto a la aplicación de partida, se va a pasar a mostrar un organigrama del funcionamiento de esta.

De forma muy general se puede resumir el funcionamiento de la siguiente forma. Se parte de un archivo XML con datos sobre las conexiones entre países, se transforma en un formato que entienda ModelNet (.graph), creando así la red general (A). Se añaden los nodos virtuales a utilizar en el experimento, obteniendo el gráfico de red definitivo (B).

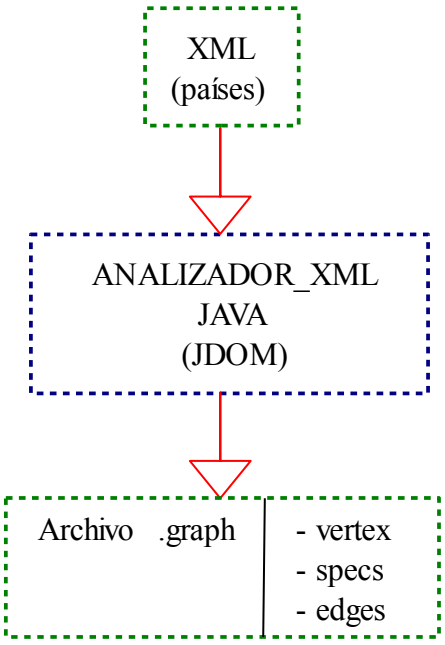
Se despliegan todos los archivos necesarios en las máquinas que participarán en las pruebas, se inicializan y se crean los scripts necesarios para llevar a cabo los diferentes experimentos (C). Por último, se ejecuta el experimento y una vez finalizado se recogen las trazas con los resultados obtenidos (D).

A continuación se mostrarán los organigramas de funcionamiento con las siguientes leyendas:

Clase Java
Archivo
Funcionamiento

AnalizadorXML

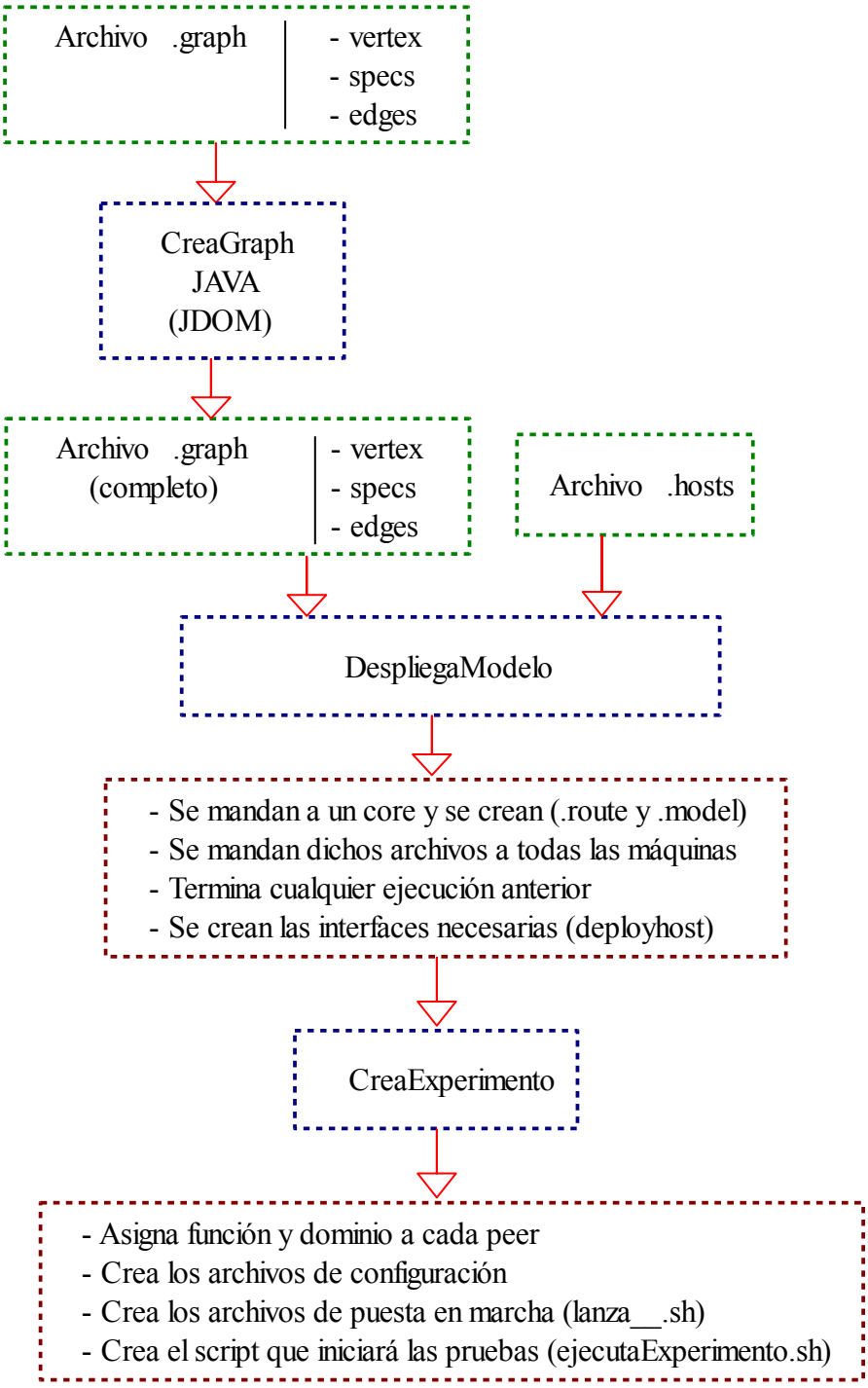
A)



ExperimentoP2PP

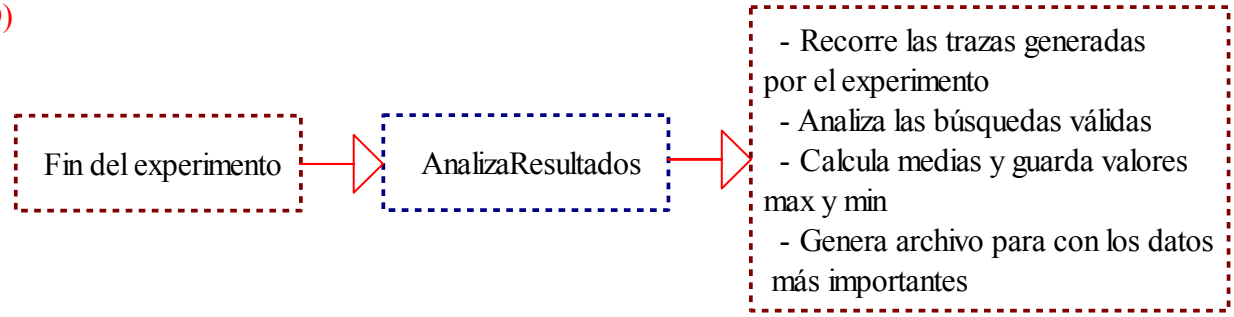
B)

C)



AnalizaResultados

D)



Una vez mostrado de forma muy general el organigrama de funcionamiento de la aplicación de partida, se va a pasar a entrar más en detalle en la función de cada parte.

3.1.1 **ANALIZADOR DEL XML**

Es la primera parte y está desarrollada en Java. Se encarga de analizar un archivo que contiene los datos del proyecto CAIDA [CAID10] para generar una topología que pueda entender ModelNet (archivo .graph).

Se parte de un archivo XML con el siguiente formato:

País origen , país destino , mínimo Round trip time , variación del Rtt , variación de retardo , paquetes perdidos

```
<SummaryReport from="Germany" to="Germany" minimumRtt="6.0" averageRtt="7.68" delayVariation="2.25" packetLoss="0.0"/>
<SummaryReport from="Germany" to="Africa" minimumRtt="93.65" averageRtt="102.64" delayVariation="3.18" packetLoss="0.01"/>
<SummaryReport from="Germany" to="Balkans" minimumRtt="68.84" averageRtt="84.02" delayVariation="5.47" packetLoss="0.29"/>
<SummaryReport from="Germany" to="France" minimumRtt="43.68" averageRtt="48.06" delayVariation="4.75" packetLoss="0.01"/>
<SummaryReport from="Germany" to="Australia" minimumRtt="292.6" averageRtt="310" delayVariation="11.57" packetLoss="0.01"/>
```

Haciendo uso de JDOM [JDOM10] va recorriendo dicho archivo en busca de enlaces en los que estén involucrados dos de los países seleccionados para formar parte de las pruebas. Una vez encontrado un enlace, se calcula el tiempo de tránsito a partir del Rtt medio y se introduce en el archivo de la topología para ModelNet.

Cuando se dé el caso de encontrar un enlace que vaya desde un país a sí mismo, se genera una entrada dentro del apartado *specs* de la topología que será usado para marcar el tiempo de tránsito entre los edge node asignados a ese país y su emulador.

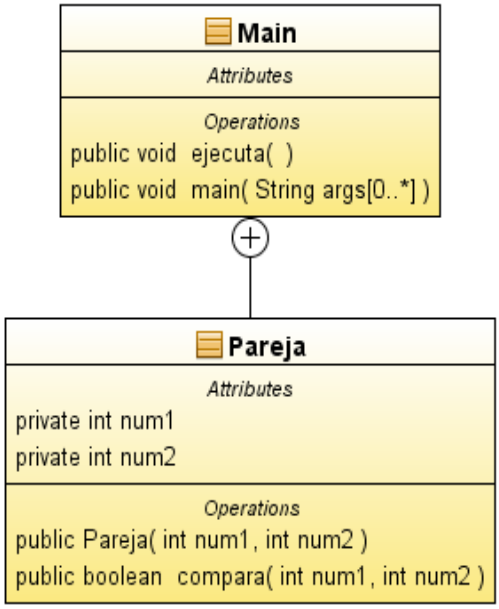


Figura 3.3: Esquema de clases de AnalizadorXML_CAIDA

La clase Main es la encargada de todo el proceso de análisis y generación del archivo de la topología. Para ello usa la clase Pareja, que básicamente guarda los números de un enlace con el fin de evitar que se introduzca más de un enlace entre dos nodos en nuestra topología. El archivo resultante es la topología central de nuestra red.

3.1.2 GENERADOR DEL EXPERIMENTO

En cada experimento se lanzan cientos de pruebas, y en cada prueba se ejecuta la implementación cientos de veces y se realizan miles de consultas. Es por ello que se creó la aplicación "ExperimentoP2PP", encargada de generar automáticamente todos los archivos necesarios para ejecutar cada experimento.

.graph → **CreaGraph** → **.graph (completo)**
(vertex / specs / edges) — JAVA (JDOM) — (vertex / specs / edges)

.graph (completo) → **DespliegaModelo**
(vertex / specs / edges) — (Despliega toda la red Modelnet)

CreaExperimento (Genera todo los archivos necesarios para la ejecución de las pruebas)

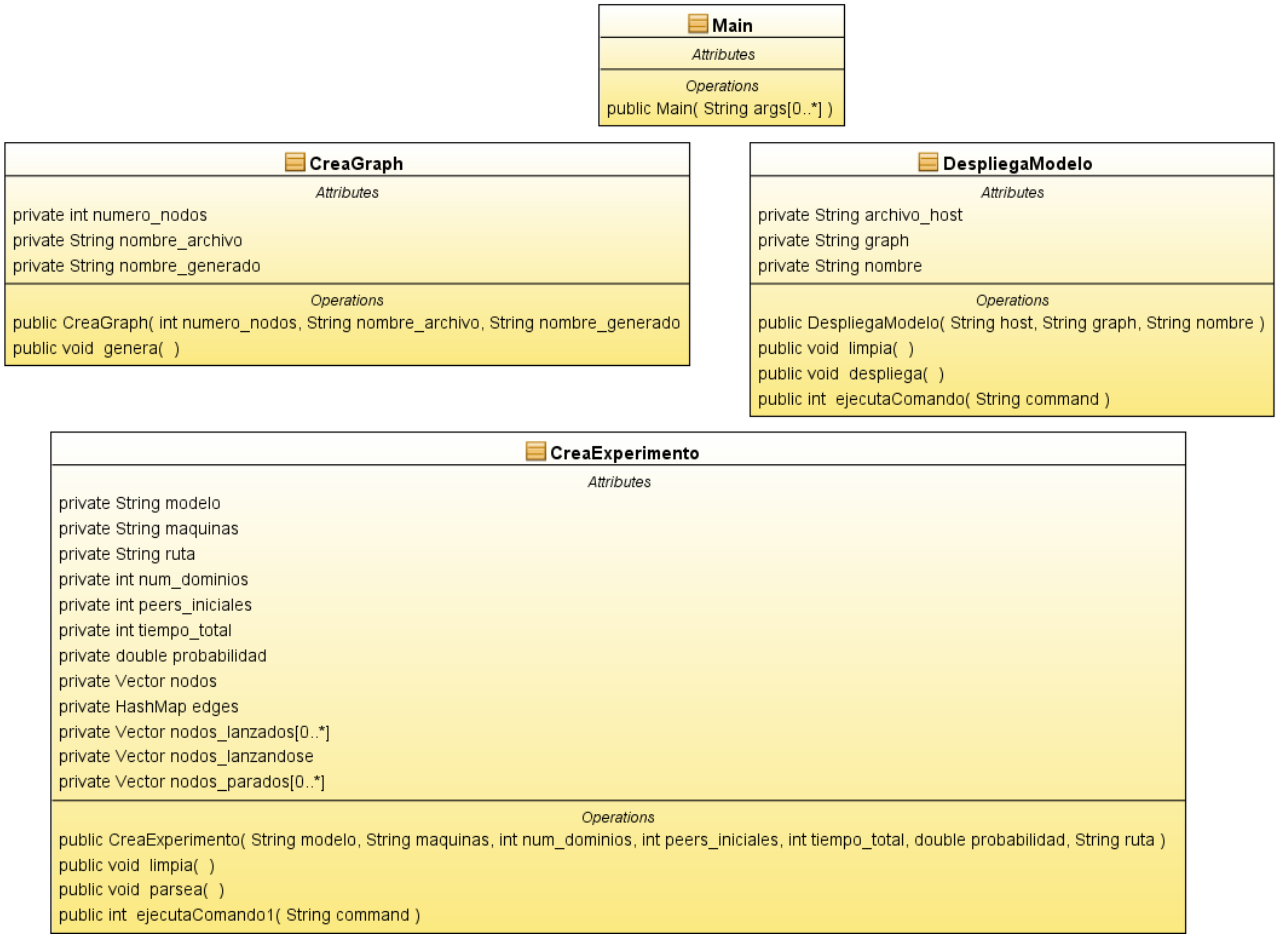


Figura 3.4: Esquema de clases de ExperimentoP2PP

En conjunto, cuando se ejecuta el generador de experimentos, la clase Main se encarga de usar las clases CreaGraph y DespliegaModelo en un primer momento. Posteriormente va llamando a la clase CreaExperimento tantas veces como pruebas se van a lanzar y va generando automáticamente un script que luego lance todas las pruebas. Una vez terminado todo el proceso de creación de archivos, el programa los comprime y manda dicho archivo comprimido a la máquina virtual que actúa como control.

La clase **CreaGraph** es la encargada de terminar el archivo con la topología de red. Esta clase recibe un archivo con la topología central de la red y le va agregando tantos nodos virtuales como se vayan a necesitar para los experimentos. Al igual que en el AnalizadorXML, para realizar esta función, hace uso de JDOM para leer el archivo de entrada, comprobando el número de enlaces y nodos existentes, para posteriormente añadir los nodos virtuales que se necesiten.

La clase **DespliegaModelo** genera tres scripts (despliegaModelnet.sh, ejecuta1.sh y ejecutaModelnet.sh).

1. DespliegaModelnet.sh fue diseñado para que se ejecutara desde la máquina virtual de control. Se encarga de mandar a un emulador el archivo con el gráfico de la topología (.graph) y el que contiene las máquinas que van a ser usadas (.hosts), para poder así generar el archivo con el modelo (.model) y el archivo con las rutas que usará ModelNet (.route). Después, los manda a todas las máquinas que participan en el experimento. Al mismo tiempo y para aumentar el rendimiento, también se encarga de terminar cualquier ejecución del experimento que haya en curso. El script creado tiene la siguiente forma (a partir de ahora, los ejemplos de scripts se limitarán a una prueba en la que sólo participará el core node *Biogridnet6* y el edge node *Edge3*).

```
#!/bin/bash

ssh root@biogridnet6.mnuc3m.es "pkill p2pmain"

ssh root@edge3.mnuc3m.es "pkill p2pmain"

scp/home/control/p2pp/experimentoP2PP/probando.graph root@biogridnet6.mnuc3m.es:/usr/modelos/probando.graph

scp/home/control/p2pp/experimentoP2PP/sample.hosts root@biogridnet6.mnuc3m.es:/usr/modelos/sample.hosts

ssh root@biogridnet6.mnuc3m.es "allpairs /usr/modelos/probando.graph > /usr/modelos/probando.route"

ssh root@biogridnet6.mnuc3m.es "mkmodel /usr/modelos/probando.graph /usr/modelos/sample.hosts >
/usr/modelos/probando.model"

scp root@biogridnet6.mnuc3m.es:/usr/modelos/probando.model
/home/control/p2pp/experimentoP2PP/probando.model

scp root@biogridnet6.mnuc3m.es:/usr/modelos/probando.route /home/control/p2pp/experimentoP2PP/probando.route

scp /home/control/p2pp/experimentoP2PP/probando.model
root@biogridnet6.mnuc3m.es:/usr/modelos/probando.model

scp /home/control/p2pp/experimentoP2PP/probando.route root@biogridnet6.mnuc3m.es:/usr/modelos/probando.route

scp /home/control/p2pp/experimentoP2PP/probando.model
root@biogridnet6.mnuc3m.es:/usr/modelos/probando.model

scp /home/control/p2pp/experimentoP2PP/probando.model root@edge3.mnuc3m.es:/usr/modelos/probando.model

scp /home/control/p2pp/experimentoP2PP/probando.route root@edge3.mnuc3m.es:/usr/modelos/probando.route
```

2. Ejecuta1.sh se encarga de mandar a la máquina virtual de control los archivos de ModelNet y de ejecutar el script creado anteriormente (despliegaModelnet.sh). También se encarga de traer a la máquina en la que se está ejecutando el generador de experimentos, el archivo con el modelo creado por ModelNet (.model), que será usado posteriormente por la clase CreaExperimento. Este script tiene la siguiente forma:

```
#!/bin/bash
ssh control@163.117.140.36 "~/p2pp/reiniciaRed.sh"
sleep 3
ssh control@163.117.140.36 "mkdir /home/control/p2pp/experimentoP2PP/"
ssh control@163.117.140.36 "mkdir /home/control/p2pp/experimentoP2PP/logs"
ssh control@163.117.140.36 "rm /home/control/p2pp/experimentoP2PP/*"
ssh control@163.117.140.36 "rm /home/control/p2pp/experimentoP2PP/logs/*"
ssh control@163.117.140.36 "pkill despliega.sh"
scp despliegaModelnet.sh control@163.117.140.36:/home/control/p2pp/experimentoP2PP/despliegaModelnet.sh
ssh control@163.117.140.36 "chmod 777 /home/control/p2pp/experimentoP2PP/despliegaModelnet.sh"
scp probando1.graph control@163.117.140.36:/home/control/p2pp/experimentoP2PP/probando.graph
scp sample.hosts control@163.117.140.36:/home/control/p2pp/experimentoP2PP/sample.hosts
ssh control@163.117.140.36 "/home/control/p2pp/experimentoP2PP/despliegaModelnet.sh"
scp control@163.117.140.36:/home/control/p2pp/experimentoP2PP/probando1.model probando.model
```

3. EjecutaModelnet.sh es el encargado de arrancar los emuladores antes de empezar cada prueba. En primer lugar, termina los procesos lanzados por las pruebas anteriores y después ejecuta el comando *deployhost* de ModelNet en todos los emuladores y edge nodes para que se creen las interfaces en las máquinas y ModelNet pueda enrutar bien los paquetes durante los experimentos. Tiene la siguiente forma:

```
#!/bin/bash

ssh root@biogridnet6.mnuc3m.es “pkill p2pmain”

ssh root@biogridnet6.mnuc3m.es “pkill checker”

ssh root@edge3.mnuc3m.es “pkill p2pmain”

ssh root@edge3.mnuc3m.es “pkill checker”

sleep 0.5

ssh root@biogridnet6.mnuc3m.es “deployhost /root/modelos/probando.model /root/modelos/probando.route”

ssh root@edge3.mnuc3m.es  “deployhost /root/modelos/probando.model /root/modelos/probando.route”
```

Finalmente, se utiliza la clase **CreaExperimento** que es el centro de esta aplicación y la encargada de crear todos los archivos que necesita la implementación de P2PP, así como los scripts para ejecutarla.

En primer lugar, se ejecuta el método parsea() tantas veces como pruebas se vayan a lanzar en el experimento. Recorre el archivo con el modelo creado por ModelNet llenando un *Vector* con los datos de todos los nodos virtuales, para a su vez almacenarlos en un objeto de la clase NodoVirtual.

El segundo paso es asignar a cada nodo virtual la función que va a tener en la prueba. Para ello se asignan a los primeros la función de checker, luego los boostrap, superpeer y finalmente los que actuarán como peers normales. En este momento también se asigna a cada nodo el dominio en el que va a estar. Una vez hecho esto, se tienen todos los datos de los nodos participantes en la prueba; por tanto, se procede a crear los archivos de configuración necesarios.

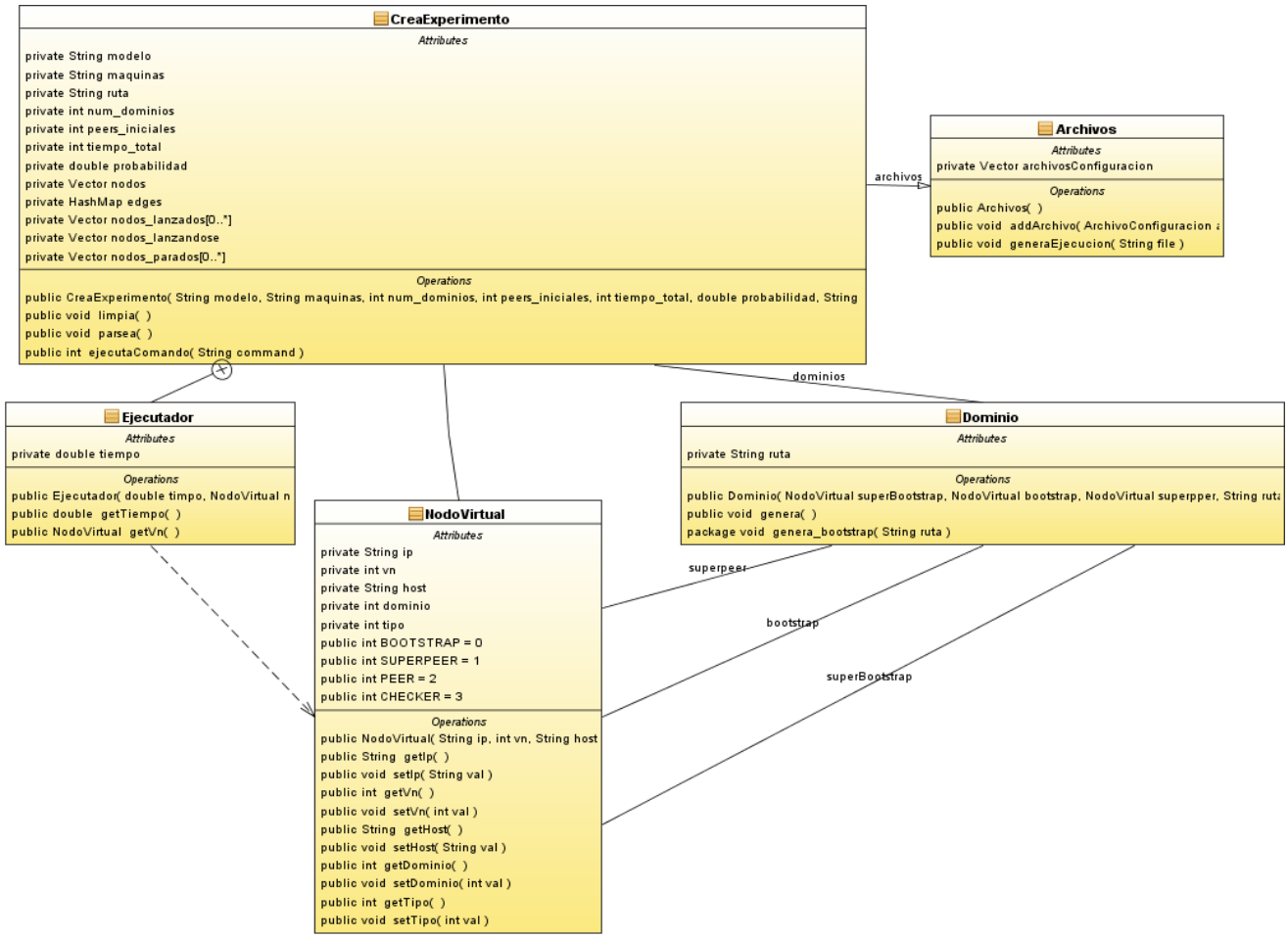


Figura 3.5: Esquema de clases de CreaExperimento

Se crean cuatro tipos de archivos de configuración con la clase *Dominio*. Los archivos de configuración de los peers configpX.txt (donde la X es el número de peer), se crean a partir de los datos del superpeer y el bootstrap del dominio al que pertenezca. Los de los superpeers configsuperY.txt (donde la Y es el número de dominio, ya que hay un superpeer por dominio), a partir de los datos del bootstrap de su dominio y del bootstrap de la red de superpeers. También en este momento se generan los archivos necesarios para la configuración de los bootstrap y los checker.

Con los archivos de configuración creados, se pasa a generar los scripts para ejecutar la prueba. Para cada edge node se generarán cuatro archivos (bootstrap, superpeers, peers y querys). Los nodos participantes en el experimento son elegidos aleatoriamente.

Hay que destacar que para la creación de las querys se usan unas distribuciones estadísticas que nos indican cuando se producirá el siguiente evento. Para la generación de los tiempos en que se producen las querys se usa una distribución de Poisson y para los tiempos de entrada y salida de los peers se usa una distribución Binomial (r = 17 ; p = 0.127) como se indica en [SENB07b].

Cuando se produce un evento de entrada o salida de un peer, se elige aleatoriamente de entre todos los peers disponibles en ese momento. Por otra parte, cuando se llega al tiempo en el que se debe hacer una query, se elige un peer desde el que comenzar la búsqueda, y luego se elige el peer a buscar dentro o fuera de su dominio dependiendo de la probabilidad marcada.

Los archivos generados tienen un formato similar al siguiente:

```
#!/bin/bash

sleep 2.468888888888889

nohup vnrunhost 12 /root/modelos/probando.model /root/p2pp/checker -a 10.0.2.196 -p 7080 -u nodo423@dominio4 -n 6004 -o nodo540@dominio3 >> /root/p2pp/logs_checker/log_querys3a4.txt &

sleep 3.037777777777778

nohup vnrunhost 12 /root/modelos/probando.model /root/p2pp/checker -a 10.0.2.41 -p 7080 -u nodo511@dominio4 -n 6012 -o nodo327@dominio1 >> /root/p2pp/logs_checker/log_querys1a4.txt &

sleep 4.008888888888887

nohup vnrunhost 12 /root/modelos/probando.model /root/p2pp/checker -a 10.0.1.34 -p 7080 -u nodo304@dominio4 -n 6020 -o nodo269@dominio4 >> /root/p2pp/logs_checker/log_querys4a4.txt &
```

Una vez generados los archivos que llevarán a cabo la ejecución de la prueba, se pasa a crear el script que recupere los resultados obtenidos (recoge.sh). Transfiere desde los edge nodes hasta la máquina virtual de control las trazas obtenidas en la prueba. Este archivo tiene el siguiente formato:

```
#!/bin/bash

scp root@edge3.mnuc3m.es:/root/Resultados.txt
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/resultados/temp.txt

cat /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/resultados/temp.txt >>
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/resultados/Resultados.txt

scp root@edge3.mnuc3m.es:/root/*.txt
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/recoge
```

Además de todos los anteriores archivos, también se crea otro script diseñado para ser ejecutado en control que se encarga de copiar los archivos a los edge nodes y ejecutar la prueba. También borra los archivos de pruebas anteriores de los edge node y mata todos los procesos relacionados con anteriores pruebas para dejar las máquinas listas para ser usadas. Después copia los archivos generados y finalmente va ejecutando los scripts creados para lanzar los bootstrap, superpeers, peers normales y las querys. Tiene el siguiente formato:

```
#!/bin/bash
/home/control/p2pp/reiniciaRed.sh
ssh root@edge3.mnuc3m.es “pkill lanza“
ssh root@edge3.mnuc3m.es “pkill p2pmain“
ssh root@edge3.mnuc3m.es “pkill -f checker“
ssh root@edge3.mnuc3m.es “rm /root/p2pp/*”
ssh root@edge3.mnuc3m.es “rm /root/*,txt”
ssh root@edge3.mnuc3m.es “rm /root/nodo*”
ssh root@edge3.mnuc3m.es “rm /root/p2pp/logs_checker/*2
ssh root@edge3.mnuc3m.es “mkdir /root/p2pp/logs_checkers”
scp /home/control/p2pp/experimentoP2PP/experimentos/* root@edge3.mnuc3m.es:/root*p2pp/
scp /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/archivos/*
root@edge3.mnuc3m.es:/root/p2pp/
ssh root@edge3.mnuc3m.es “chmod 777 /root/p2pp/*.sh”
ssh root@edge3.mnuc3m.es “/root/p2pp/lanzabootedge3.sh”
```

El último archivo generado por esta herramienta (ejecutaExperimento.sh) lo crea la clase Main a medida que se van creando las diferentes pruebas. Es el encargado de ejecutar desde control los scripts ejecutaRemoto.sh y recoge.sh. Es el único que se debe ejecutar desde control para la realización de un experimento completo.

```
#!/bin/bash
pkill ejecutaRemoto.sh
chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnetprobando5.sh
/home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnetprobando5.sh
sleep 15
echo "Empieza el experimento"
date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt
echo "EXPERIMENTO: dom5 prob:0.2 indice:0 empezamos." >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt
chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/archivos/ejecutaRemoto.sh
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/archivos/ejecutaRemoto.sh
sleep 3060
date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt
echo "EXPERIMENTO: dom5 prob:0.2 indice:0 completado, recogemos" >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt
chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/recoge.sh
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice0/recoge.sh
echo "Información recogida"
sleep 60
pkill ejecutaRemoto.sh
chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnetprobando5.sh
/home/control/p2pp/experimentoP2PP/experimentos/ejecutaModelnetprobando5.sh
sleep 15
echo "Empieza el experimento"
date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt
echo "EXPERIMENTO: dom5 prob:0.2 indice:1 empezamos." >>
/home/control/p2pp/experimentoP2PP/experimentos/log.txt
chmod 777 /home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice1/archivos/ejecutaRemoto.sh
/home/control/p2pp/experimentoP2PP/experimentos/dom5_prob0.2_indice1/archivos/ejecutaRemoto.sh
sleep 3060
date >> /home/control/p2pp/experimentoP2PP/experimentos/log.txt
```

3.1.3 **ANALIZADOR DE RESULTADOS**

Finalizado el experimento → **analizaResultados**
(Recoge trazas y las analiza)

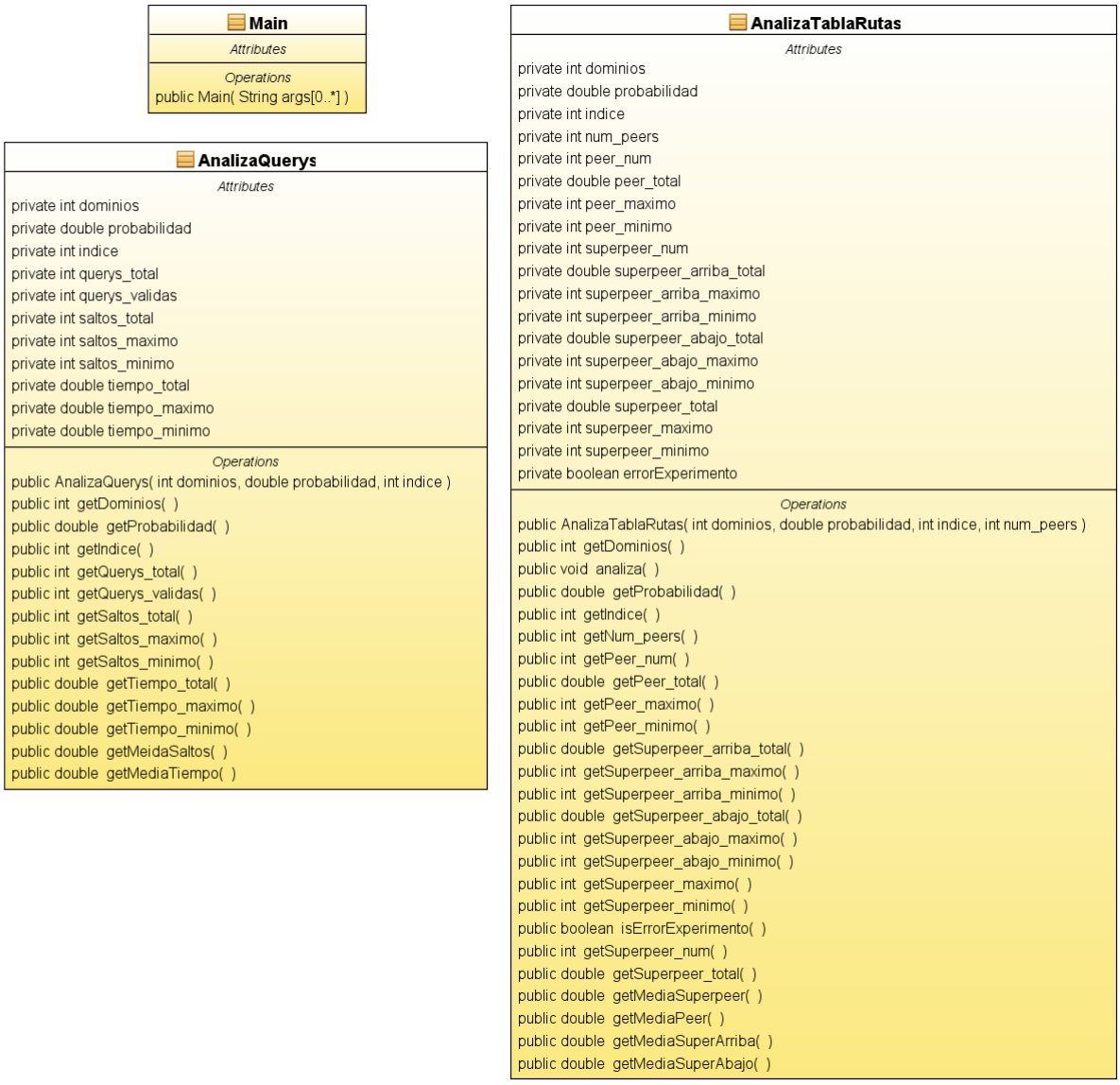


Figura 3.6: Esquema de clases de AnalizadorResultados

Es la última aplicación y la encargada de obtener las trazas generadas durante el experimento. Se compone de dos clases Java que parsean dichas trazas y calculan sus medias.

La clase **Main** recorre las diferentes pruebas del experimento y realiza los cálculos pertinentes usando las otras dos clases (AnalizaQuerys y AnalizaTablaRutas). Una vez finalizado todo el proceso, guarda la información en un archivo compatible con Matlab, que está formado por los siguientes valores:

Número_de_dominios, probabilidad, Media_saltos, Saltos_máximos, Saltos_mínimos, Media_Tiempo, Tiempo_máximo, Tiempo_mínimo, Media_tabla_Superpeer, Máximo_tabla_Superpeer, Media_tabla_Peer, Máximo_tabla_Peer, Media_tabla_Super_arriba, Máximo_tabla_Super_arriba, Media_tabla_Super_abajo, Máximo_tabla_Super_abajo

La clase **AnalizaQuerys** recorre el archivo generado por el checker para calcular los datos relacionados con la red y tiene el siguiente formato:

Número de secuencia , peer buscado , código , saltos, peer buscador, IP buscador , tiempo , IP buscado

```
67, nodo224@dominio0, 200, 4, nodo556@dominio3, 10.0.2.198:7080, 9.294453, 10.0.1.157:7080

83, nodo298@dominio2, 200, 4, nodo474@dominio1, 10.0.3.188:7080, 6.702421, 10.0.3.166:7080

99, nodo81@dominio0, 200, 5, nodo128@dominio3, 10.0.2.16:7080, 3.364065, 10.0.1.11:7080

107, nodo316@dominio0, 200, 4, nodo431@dominio3, 10.0.2.54:7080, 0.818637, 10.0.0.169:7080

123, nodo72@dominio3, 200, 4, nodo51@dominio2, 10.0.2.7:7080, 1.686098, 10.0.0.138:7080

131, nodo553@dominio0, 200, 4, nodo167@dominio1, 10.0.1.22:7080, 0.740008, 10.0.1.70:7080

139, nodo45@dominio3, 200, 6, nodo469@dominio2, 10.0.1.59:7080, 3.65577, 10.0.3.5:7080

147, nodo42@dominio2, 200, 4, nodo454@dominio1, 10.0.1.185:7080, 0.568872, 10.0.1.134:7080

163, nodo184@dominio0, 200, 5, nodo328@dominio3, 10.0.2.169:7080, 1.733433, 10.0.1.152:7080

203, nodo87@dominio3, 200, 5, nodo133@dominio1, 10.0.0.146:7080, 0.680168, 10.0.0.12:7080

219, nodo374@dominio3, 200, 5, nodo254@dominio0, 10.0.1.160:7080, 1.806811, 10.0.1.175:7080
```

Una vez mostrado un ejemplo, cabe destacar los términos tercero, cuarto y séptimo que significan lo siguiente. El tercero es el código de respuesta, si es 200 la búsqueda ha sido satisfactoria y tenemos que analizar dicha entrada. El cuarto son los saltos necesarios para encontrar el destino buscado y el séptimo es el tiempo utilizado en dicha búsqueda. El programa va sumando estos datos y finalmente calcula la media guardando además los valores máximo y mínimo.

Por último, la clase **AnalizaTablaRutas** busca la información en las trazas generadas por los peers. Lo que nos interesa de cada peer es su número medio, máximo y mínimo de entradas para posteriormente calcular la media del experimento. El formato es el siguiente:

Tipo peer , número de entradas en su tabla de rutas

```
NORMALPEER-total → TablaRutas=[28]

NORMALPEER-total → TablaRutas=[28]

NORMALPEER-total → TablaRutas=[30]

NORMALPEER-total → TablaRutas=[32]

NORMALPEER-total → TablaRutas=[34]
```

4 Versión mejorada de la aplicación de partida

4.1 Inconvenientes de aplicación de partida

Primeramente hay que decir que los inconvenientes que se van a expresar en este apartado lo son, debido a que los objetivos buscados no son exactamente iguales entre las dos aplicaciones. Se van a destacar las decisiones particulares que se tomaron a la hora de diseñar la aplicación de partida y que por las características del proyecto que se está explicando en este documento se van a cambiar para poder así obtener una solución más general, pero basándose siempre en los principios y programas más importantes de la aplicación desde la que se parte.

A la hora de diseñar el nuevo proyecto se han tenido en cuenta gran variedad opciones, y es probable que no siempre se haya tomado la decisión más general, pero cabe decir que conseguir esto es muy complicado debido a que existen muchos puntos de vista y es casi imposible realizar una aplicación 100% generalista. No obstante, se ha conseguido diseñar una aplicación lo suficientemente general como para que cualquier investigador o programador sea capaz de usar todo el potencial de la aplicación de partida pero con una complejidad muy inferior a la inicial, dotando así a la nueva aplicación de mayor atractivo y versatilidad a la hora de poder ser utilizada para desarrollar y probar nuevos protocolos, programas P2P, etc, sin la necesidad poseer un entorno real con todos los inconvenientes y costes que ello conllevaría.

Por todo lo contado anteriormente, se va a intentar explicar cada uno de los cambios y mejoras realizadas en esta segunda versión respecto a la primera, así como las decisiones tomadas que varíen entre ambas, pero intentando no entrar mucho en el detalle en cuanto al código de las aplicaciones se refiere si no es estrictamente necesario, para evitar la posible confusión del lector.

La forma de abordar este apartado y los siguientes será, ir comentando las diferencias entre versiones siguiendo el orden en el que se explicó la aplicación de partida en el apartado anterior.

4.2 Requisitos

En esta sección se va a comentar qué requisitos o criterios se han seguido para diseñar la aplicación, para que posteriormente en los apartados siguientes se pueda explicar tanto los cambios realizados en base a los requisitos, así como implementación llevada a cabo. Durante estos capítulos se expondrá la forma con la que estaba diseñada la primera versión y el diseño de la segunda para que se pueda ver del mejor modo posible todas las diferencias entre ambos proyectos, pero siempre sin olvidar el punto de partida y el objetivo final de ambas.

El primer cambio realizado para conseguir la generalización, que aún no siendo significativo hay que mencionar, es el cambio de nombres en las clases que pertenecen al proyecto debido a la variación de alguna de sus funciones, pasando por tanto a ser clases diferentes, quedando de la siguiente manera:

APLICACIÓN DE PARTIDA		APLICACIÓN DESARROLLADA
CreaGraph	---------->	Estructura
DespliegaModelo	---------->	Instalar
---------	---------->	Ejecución
CreaExperimento	---------->	CreaExperimento
AnalizaResultados	---------->	AnalizaResultados

De ahora en adelante y para resumir, cuando se refiera en este documento a la aplicación de partida se hará como versión 1.0 y cuando se refiera a la aplicación diseñada para este proyecto, se hará como versión 2.0.

A la hora de especificar los datos necesarios para la configuración, instalación y lanzamiento de los experimentos, en la versión 1.0 se hacía directamente por línea de comandos a la hora de ejecutar la aplicación Java pero en esta versión, todo lo necesario, deberá ir expresado donde corresponda en el XML habilitado para tal fin. En la siguiente página se muestra un ejemplo de como sería el XML para un dominio y para dos tipos de nodos (bootstrap y superpeers), debido a que para más dominios y distintos tipos será similar pero más extenso.

Este archivo XML es sin duda el elemento más importante de esta versión 2.0, debido a que si está mal configurado nada de lo que viene después se ejecutará bien, en el caso de que se ejecute. En este apartado se van a comentar las partes globales que tiene el archivo sin entrar en detalle de cada etiqueta, ya que eso se hará en el apartado 4.6 donde se explicará la función de cada etiqueta. En él hay que diferenciar cinco partes, que son: topología de red, definición de los dominios, esquema de directorios, aplicaciones y experimentos.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE proyecto SYSTEM "/home/fgomez/PFC2/ficheros/projectP2PP.dtd">
<proyecto>
  <red>
    <machines ruta = "/home/fgomez/PFC2/ficheros/"> sample.hosts </machines>        <!-- Archivo .hosts-->
    <topologia ruta = "/home/fgomez/PFC2/ficheros/"> global.graph </topologia>      <!-- Archivo .graph-->
    <salida> DESPLEGADO1 </salida>                                                  <!-- Nombre de salida de los archivos creados -->
    <domain> 1 </domain>
    <num_experimentos>10</num_experimentos>
    <num_peers > 1000 </num_peers>                                                  <!--Número total experimento-->
    <num_checkers> 8 </num_checkers>
    <prob> 1 </prob>
    <tiempo_total>3000</tiempo_total>
    <archivo_general ruta = "./archivos/experimentos/"/>                            <!-- Path de archivo de experimento -->
    <nom_domain> .mnuc3m.es </nom_domain>
    <control nombre = "control">163.117.140.36</control>
  </red>
  <dominios>
    <checkers n_peers = "8"/>
    <numero id = "1" n_peers = "1000" distribucion = "random">
      <node type = "0">                                                  <!-- BOOTSTRAP y BOOTSTRAPSUPERPEERS-->
        <n_peers>2</n_peers>
        <opciones ejecucion = "$ruta1$ruta2$ruta3$ruta4$">                          <!-PARTE DE INSTALACIÓN->
          <ruta1> /root/p2pp/p2pmain </ruta1>                                       <!-PARTE DE INSTALACIÓN->
          <ruta2> /root/p2pp/configboot.txt </ruta2>                                <!-PARTE DE INSTALACIÓN->
          <ruta3> /root/p2pp/boot$node.log</ruta3>                                  <!-PARTE DE INSTALACIÓN->
          <ruta4> nodo$node </ruta4>                                                <!-PARTE DE INSTALACIÓN->
        </opciones>
      </node>
      <node type = "1">                                                  <!-- SUPERPEERS -->
        <n_peers>1</n_peers>
        <opciones ejecucion = "$ruta1$ruta2$ruta3$ruta4$">                          <!-PARTE DE INSTALACIÓN->
          <ruta1> /root/p2pp/p2pmain </ruta1>                                       <!-PARTE DE INSTALACIÓN->
          <ruta2> /root/p2pp/configsuper1.txt </ruta2>                              <!-PARTE DE INSTALACIÓN->
          <ruta3> /root/p2pp/super$node.log</ruta3>                                 <!-PARTE DE INSTALACIÓN->
          <ruta4> nodo$node@dominio1</ruta4>                                        <!-PARTE DE INSTALACIÓN->
        </opciones>
      </node>
    </numero>
  </dominios>
  <instalar>
    <inicio_mi_ruta ruta= "/home/fgomez/PFC2/MiPFC/archivos/instalar/"/>
    <inicio_mi_ruta_INFO
 ruta="/home/fgomez/PFC2/MiPFC/archivos/instalar/INFO/">INFORMACION1.txt</inicio_mi_ruta_INFO>
    <inicio_mi_ruta_admin ruta = "/home/fgomez/PFC2/MiPFC/archivos/administracion/"/>
    <inicio_mi_ruta_config ruta = "./archivos/configuracion/"/>                                 <!-PARTE DE INSTALACIÓN->
    <inicio_mi_ruta_app ruta = "./archivos/aplicaciones/"/>
    <inicio_mi_ruta_resultados ruta = "/home/fgomez/PFC2/MiPFC2/archivos/resultados/"/>

    <inicio_dir_local ruta = "/home/control/Aproject/instalar/"/>
    <inicio_dir_local_admin ruta = "/home/control/Aproject/"/>
    <inicio_dir_local_zip ruta = "/home/control/Aproject/experimentos"/>
    <inicio_dir_local_ejecucion ruta = "/home/control/Aproject/archivos/experimentos/"/>
    <inicio_dir_local_config ruta = "/home/control/Aproject/configuracion/"/>                   <!-PARTE DE INSTALACIÓN->
    <inicio_dir_local_app ruta = "/home/control/Aproject/aplicaciones/"/>

    <inicio_dir_remota ruta = "/usr/modelos/"/>
    <inicio_dir_remota_ejecucion ruta = "/root/p2pp/"/>
    <inicio_dir_remota_raiz ruta = "/root/"/>
    <inicio_file_zip ruta = "./archivos/experimentos"/>
  </instalar>
  <aplicaciones numero = "2">
    <app ruta = "./archivos/aplicaciones/" > p2pmain </app>
    <app ruta = "./archivos/aplicaciones/" > checker </app>
  </aplicaciones>
  <experimentos numero = "2">
 <edge1 ruta = "/home/fgomez/MiPFC2/archivos/configuracion/">lanza4experimentoproxmoxedge1.sh</edge1>
 <edge2 ruta = "/home/fgomez/MiPFC2/archivos/configuracion/">lanza4experimentoproxmoxedge2.sh</edge2>
  </experimentos>
</proyecto>
```

Topología de red

- Path de archivos de partida (graph y hosts).
- Número de dominios.
- Número de peers.
- Nombre del dominio.
- Nombre y dirección IP de la máquina que hará la función de control.
- Número checkers.
- Probabilidad.
- Nombre salida archivos.
- Repeticiones del experimento.
- Tiempo total del experimento.
- Path de los experimentos.

Definición de los dominios

- Número del dominio.
- Tipo de peer.
- Número de peers/dominio.
- Número peers/tipo.
- Tipo de colocación.
- Forma de ejecutar un peer.

Esquema de directorios

- Path de la máquina local.
- Path de la máquina que actúa como control.
- Path de los edge y core nodes.

Aplicaciones

Path y aplicaciones que se quieren.
ejecutar en los experimentos.

Experimentos

Path y archivos necesarios para
ejecutar los experimentos.

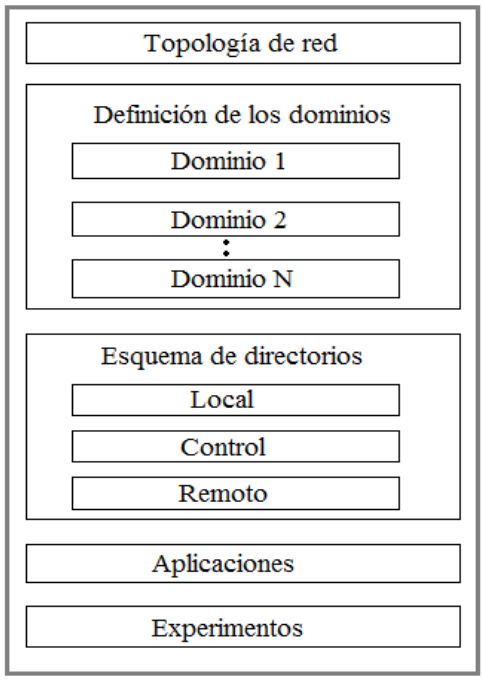


Figura 3.7: Esquema de fichero XML de configuración

4.3 **Principales cambios**

Una vez comentados los principales requisitos de esta versión 2.0, se va a entrar más en detalle de lo que se ha modificado para cumplirlos. En primer lugar, en cuanto al escenario de pruebas, comentar que a nivel de máquinas físicas, es parecido al expuesto en el apartado 3.2, con la única salvedad de que se ha añadido la máquina física carcoma y que todas las máquinas están conectadas a la red pública del laboratorio 163.117.140.0/24. Del escenario lógico explicado en ese mismo apartado cabe decir que se ha utilizado el mismo que el inicial.

La primera gran diferencia entre ambas versiones teniendo en cuenta el objetivo buscado de generalizar la aplicación, es que en este caso está formada por dos partes con objetivos individuales más precisos.

Figura 3.8: Esquema general de funcionamiento

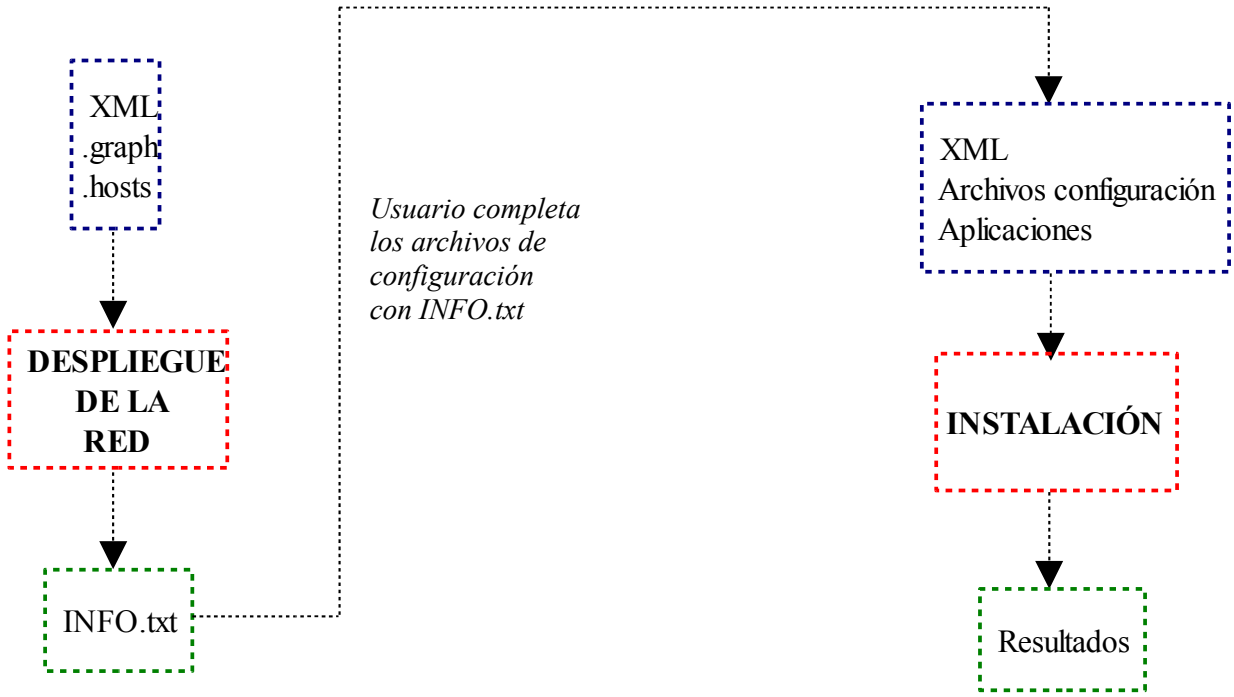
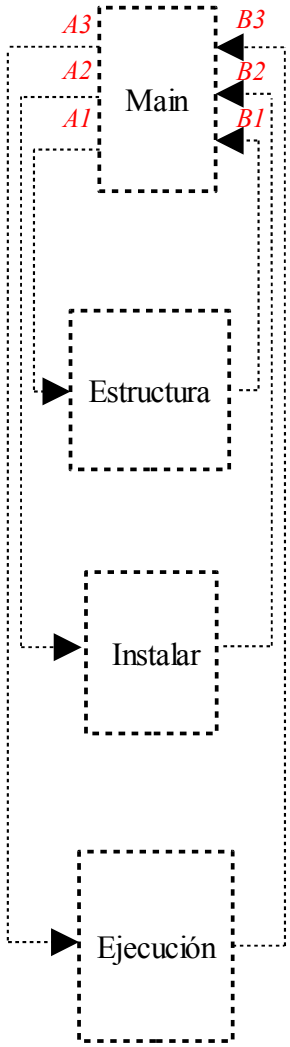


Figura 3.9: Esquema general de las partes de la aplicación

A modo general, lo que hace la aplicación es, partiendo de un XML, un archivo graph, un archivo .hosts y los archivos de configuración, instalar las aplicaciones en las 1000 máquinas virtuales que van a participar, así como ejecutar los experimentos y recoger sus resultados.

Siendo más precisos, en la parte de despliegue de la red se parte de, el archivo .graph con la topología central de red, el archivo .hosts donde se encuentran las máquinas que van a participar (edges y cores) y del archivo XML con una estructura determinada en donde se pondrán todos los parámetros necesarios para el despliegue de la red ModelNet. Finalmente, después de realizar varias funciones que más adelante se explicarán con detalle, la aplicación genera un archivo de texto con toda la información del escenario creado por esta parte. La explicación se hará en función del XML mostrado en la página 49 del apartado 4.2.



Main – Estructura (*A1*)

- Utiliza la primera parte del XML (topología de red), extrayendo el nombre del archivo con la topología central de red, el número de dominios y el nombre del archivo con la estructura de red completa.

Estructura – Main (*B1*)

- Crea el archivo .graph con la topología de red completa.

Main – Instalar (*A2*)

- Utiliza las etiquetas de la tercera parte del XML (esquema de directorios) referentes a la máquina donde se desarrolla el software y a la de control para poder instalar en ella las aplicaciones necesarias.

Instalar – Main (*B2*)

- Crea el archivo que generará el modelo (.model) y los archivos que instalarán las aplicaciones y generará las interfaces en los nodes.

Main – Ejecución (*A3*)

- Utiliza las etiquetas de la tercer parte referente a la máquina local para poder crear el archivo con toda la información de los peers.

Ejecución – Main (*B3*)

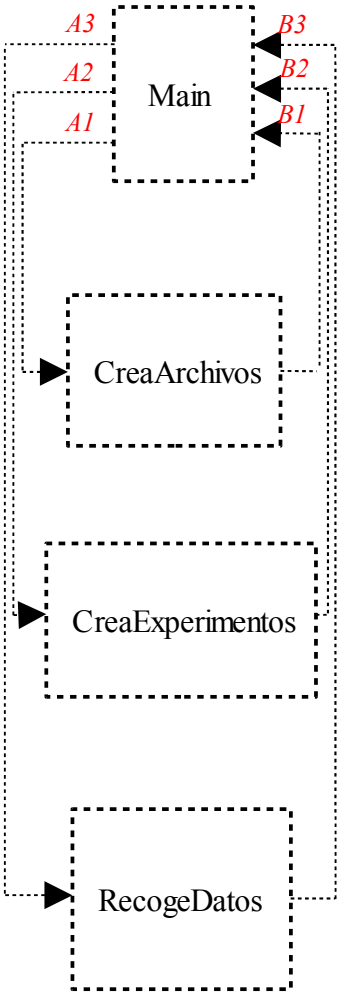
- Crea el archivo con la información de los peers (INFO.txt).

Figura 3.10: Esquema de la parte de despliegue de la red

Figura 3.11: Esquema de clases de la parte del despliegue de la red [*]

[*] No están todos los parámetros en los métodos constructores de las clases para simplificar la figura. Ver el código completo para visualizarlos.

Una vez el desarrollador ha completado los archivos de configuración de los peers, superpeers, bootstraps y checkers con la información de INFO.txt, en esta segunda parte de instalación, se parte del mismo archivo XML que en el despliegue de la red y de los archivos anteriormente mencionados. Finalmente, se crean los archivos necesarios para la ejecución del experimento, se han usado y los resultados de se encuentran en la máquina de control.



Main – CreaArchivos (*A1*)

- Extrae la información del archivo creado en la parte de despliegue de red (INFO.txt) y utiliza las etiquetas de la parte de instalación para poder generar los experimentos que serán ejecutados. También extrae la información de donde se definen los dominios para saber como ejecutar las aplicaciones.

CreaArchivos – Main (*B1*)

- Crea archivos para lanzar los bootstraps (lanza1bootedgeX.sh).
- Crea archivos para lanzar los peers (lanza2inicioedgeX.sh).
- Crea archivos para lanzar los superpeers (lanza3superedgeX.sh).
- Crea archivos para lanzar cada experimento (ejecutaRemoto.sh).
- Crea archivos para recuperar los datos (recoge.sh).

Main – CreaExperimentos (*A2*)

- Utiliza las partes del XML de definición de los dominios y esquema de directorios para crear los archivos con los experimentos.

CreaExperimentos – Main (*B2*)

- Crea archivos de experimentos (lanza4experimentoedgeX.sh).

Main – RecogeDatos (*A3*)

- Utiliza las etiquetas de la parte de los directorios donde el usuario quiere que se depositen los resultados obtenidos durante la prueba.

RecogeDatos – Main (*B3*)

- Crea el archivo que recoge los datos desde la máquina de control.

Figura 3.12: Esquema de la parte de instalación

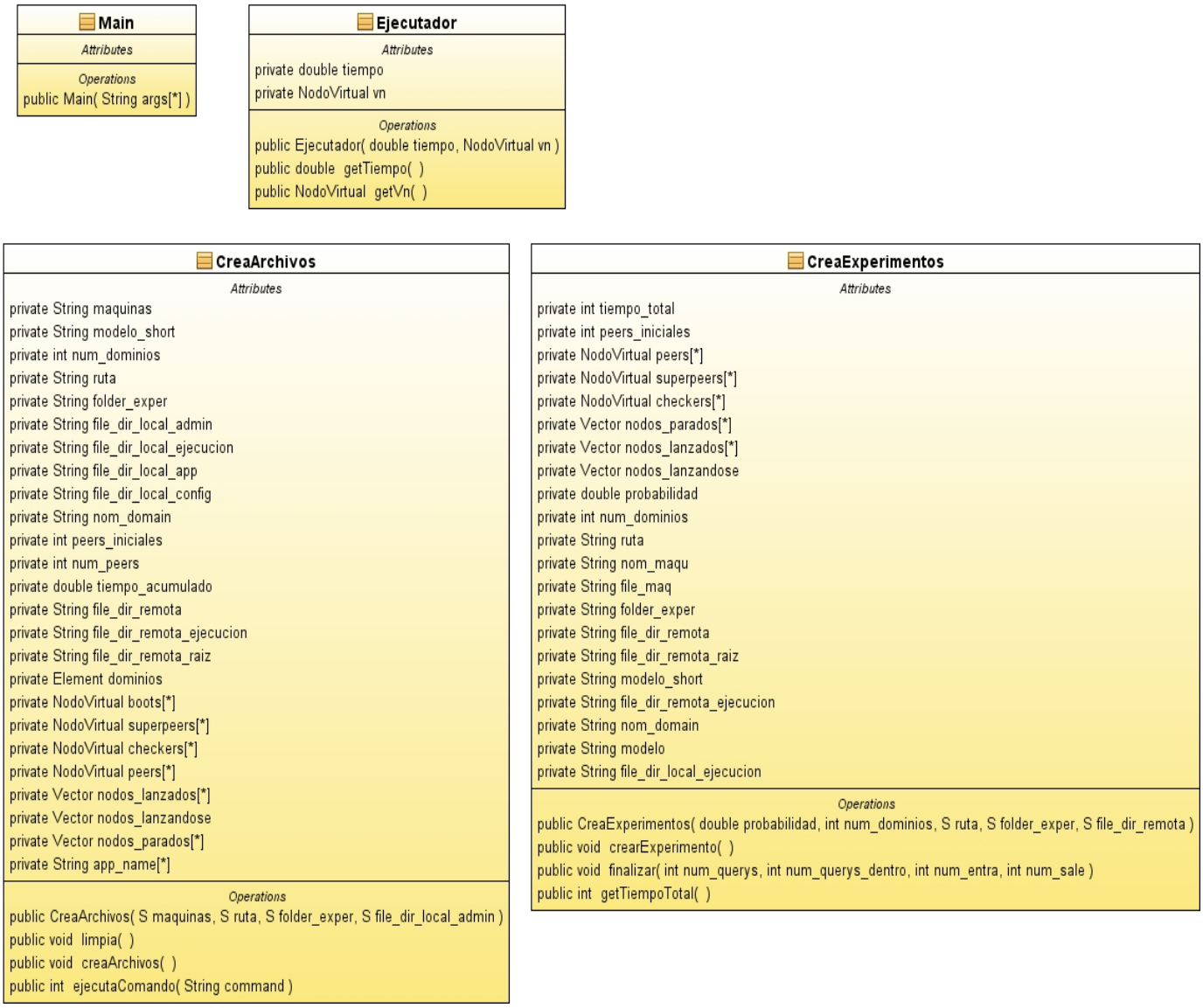


Figura 3.13: Esquema de clases de la parte de instalación [*]

Una vez los experimentos hayan concluido satisfactoriamente, se pasa a ejecutar el mismo programa para analizar los resultados que en la aplicación de partida (AnalizaResultados).

[*] No están todos los parámetros en los métodos constructores de las clases para simplificar la figura. Ver código completo para visualizarlos.

4.4 **Diseño**

Una vez explicado los requisitos y cambios generales entre ambas versiones, en este apartado y el siguiente se van a explicar con más detalle dichos cambios pero evidentemente siempre centrándonos en el proyecto que en este documento se explica, debido a que la aplicación de partida ya se explicó en el apartado 3.3.

En primer lugar y para poder hacerse una idea de todo el proceso que hay que llevar a cabo para poder ejecutar un experimento de principio a fin se va a mostrar un esquema con todos los pasos necesarios. (Se llamará máquina principal a aquella donde se ha desarrollado el software; es decir, Biogridnet2).

1. Configurar archivo XML (se necesita un archivo por cada experimento completo que se quiere realizar). En este proyecto se han usado XML5DominiosFijo, XML5DominiosAleatorio, etc.
2. Ejecutar en máquina principal la aplicación correspondiente al despliegue de la red (MiPFC).
3. Finaliza automatización del despliegue de la red.
4. Completar archivos de configuración con archivo INFO.txt.
5. Ejecutar en máquina principal la aplicación correspondiente a la instalación (MiPFC2).
6. Ir a máquina que actúa como "control" y descomprimir archivo con experimentos.
7. Dar permisos a ejecutaExperimento.sh y ejecutar ese script.
8. Finalizan todos los experimentos.
9. Ejecutar en máquina principal script Recoge_local.sh para almacenar resultados en la máquina principal.
10. Ejecutar en máquina principal el programa analizaResultados.
11. Renombrar archivos resultantes des .txt a .csv para que Matlab pueda leerlos.
12. Cargar los archivos .csv en Matlab y ejecutar los archivos para crear las gráficas.

Figura 3.14: Esquema de ejecución completa

A continuación se va a detallar cada sección de la aplicación desarrollada y las diferencias de criterios y código respecto a la aplicación de partida.

4.4.1 **DESPLIEGUE DE LA RED**

Como ya se comentó anteriormente, los primeros cambios que se pueden apreciar son en los nombres de las clases que participan, ya que en algunos casos sus funciones no son exactamente iguales.

CreaGraph ----------> Estructura
Despliegamodelo----------> Instalar
---------- ----------> Ejecución

La dinámica que se va a seguir para explicar los cambios va ser la siguiente: En la columna de la izquierda se explicará la versión de partida (versión 1.0) y en la de la derecha la nueva aplicación (versión 2.0)

CreaGraph VERSIÓN 1.0 (120 líneas)	**Estructura** VERSIÓN 2.0 (207 líneas)
1. Añade número de nodos virtuales a .graph. Dependiendo si la distribución es "fija" o "aleatoria" se colocan los peers dentro de los dominios.	**1.** Igual, pero para realizar esta función ahora se utiliza JDOM para leer la información que está contenida en el archivo XML que ha configurado previamente el desarrollador.
2. Si el nº de dominios es cero o mayor que el nº de países, el nº de dominios elegido para ese experimento será el nº de países.	
3. Se distribuyen los nodos virtuales usando el módulo; es decir, para 3 dominios y 5 peers: - Peer 1 ---------> Dominio 1 - Peer 2 ---------> Dominio 2 - Peer 3 ---------> Dominio 3 - Peer 4 ---------> Dominio 1 - Peer 5 ---------> Dominio 2	**3.** Ahora como se usa JDOM para extraer la información del XML la distribución depende del nº de peers que el desarrollador ponga en cada dominio y del tipo de distribución que desee, hay dos casos: - Fijo: Dom1=50 peers, Dom 2=50 peers Los 50 primeros serán del dominio 1 y del 51 al 101 del dominio 2. - Random: Se hace una función aleatoria. Se colocan peers en todos los países.
4. De una vez se creaban los experimentos para todos los dominios correspondientes, por ejemplo, 1,5,10 y 20.	**4.** Con la introducción del XML, sólo existe uno para cada nº de dominios (por ejemplo 5), por tanto, una ejecución completa generará todos los experimentos necesarios para ese nº de dominios.

DespliegaModelo 1.0 (189 líneas)

1. Genera los tres scripts necesarios para poder tener todos las nodos virtuales que participarán en la prueba listos para ser usados. Estos scripts son despliegaModelnet.sh, ejecuta1.sh y ejecutaModelnet.sh.

-DespliegaModelnet.sh: Genera los archivos model y .route a partir del .graph y el .hosts para que Modelnet pueda encaminar bien los paquetes. Además los envía a las máquinas.

- Ejecuta1.sh: Manda los archivos necesarios desde la máquina principal a control para que el script anterior pueda ejecutarse desde la máquina principal a control. Además manda el archivo .model desde control a la máquina principal.

- EjecutaModelnet.sh: Arranca los emuladores que participarán en el experimento y con el comando *deployhost* crea las interfaces necesarias.

Instalar 2.0 (288 líneas)

1. Realiza básicamente la misma función con algunos pequeños cambios, pero la gran diferencia está en el concepto, debido a que todas los directorios, aplicaciones, archivos a mandar, máquinas a utilizar, direcciones IPs, etc, se leen mediante JDOM a partir del XML que previamente ha tenido que configurar bien el desarrollador.

A continuación y debido a una de las mayores diferencias entre ambas versiones, se ha creado una nueva clase Java que no existía en la primera versión y que aparece debido a la necesidad de mostrar la desarrollador toda la información referente al escenario creado por ModelNet para que pueda crear adecuadamente sus archivos que configurarán los peers con sus funciones determinadas. Esta generalización es necesaria para usar la aplicación para cualquier programa y no sólo el P2PP jerárquico como hasta ahora.

Ejecución 2.0 (343 líneas)

En la aplicación de partida no existe. Se ha creado para lograr generalizar el programa en base a que cada experimento pueda tener una serie de diferencias respecto a otro, tales como la IP de los peers, número de peers por cada dominio, etc. Esta es la última clase que se usa en el despliegue de la red y su función es la de mostrar al desarrollador toda la información referente a los peers del experimento que se van a ejecutar, con el fin de que en un paso intermedio entre el despliegue de la red y la instalación pueda crear los archivos de configuración pertinentes. Un ejemplo para el caso en que se tenga 10 dominios y 100 peer sería (sólo se mostrarán los 2 primeros peers de cada dominio):

Ejemplo de archivo INFO.txt

```
--- Bootstrap ---
TIPO/VN/IP/EDGE/DOMINIO/
0/0/10.0.3.129/edge9/-1/
0/1/10.0.0.1/edge2/1/
0/399/10.0.2.178/edge7/2/
0/451/10.0.0.185/edge3/3/
0/486/10.0.2.61/edge6/4/
0/492/10.0.1.62/edge4/5/
0/575/10.0.1.73/edge4/6/
0/647/10.0.3.209/edge9/7/
0/917/10.0.2.243/edge7/8/
0/934/10.0.3.117/edge8/9/
0/977/10.0.0.251/edge3/10/

--- Superpeers ---
TIPO/VN/IP/EDGE/DOMINIO/
1/2/10.0.0.129/edge3/1/
1/400/10.0.3.51/edge8/2/
1/452/10.0.1.57/edge4/3/
1/487/10.0.2.189/edge7/4/
1/493/10.0.3.190/edge9/5/
1/576/10.0.1.200/edge5/6/
1/648/10.0.0.82/edge2/7/
1/918/10.0.3.115/edge8/8/
1/935/10.0.3.245/edge9/9/
1/978/10.0.1.123/edge4/10/

--- Peers---
TIPO/VN/IP/EDGE/DOMINIO/
2/3/10.0.1.1/edge4/1/
2/4/10.0.1.129/edge5/1/
2/103/10.0.1.142/edge5/2/
2/104/10.0.2.13/edge6/2/
2/203/10.0.0.27/edge2/3/
2/201/10.0.1.16/edge7/3/
2/303/10.0.2.166/edge7/4/
2/304/10.0.3.39/edge8/4/
2/403/10.0.0.179/edge3/5/
2/404/10.0.1.51/edge4/5/
2/503/10.0.0.192/edge3/6/
2/504/10.0.1.64/edge4/6/
2/603/10.0.3.76/edge8/7/
2/604/10.0.3.204/edge9/7/
2/703/10.0.3.216/edge9/8/
2/704/10.0.0.89/edge2/8/
2/803/10.0.1.229/edge5/9/
2/804/10.0.2.101/edge6/9/
2/903/10.0.3.241/edge9/10/
2/904/10.0.0.114/edge2/10/

--- Checkers ---
TIPO/VN/IP/EDGE/DOMINIO/
3/398/10.0.2.50/edge6/-1/
3/450/10.0.0.57/edge2/-1/
3/485/10.0.1.190/edge5/-1/
3/491/10.0.0.190/edge3/-1/
3/574/10.0.0.201/edge3/-1/
3/646/10.0.3.81/edge8/-1/
3/916/10.0.2.115/edge6/-1/
3/933/10.0.2.245/edge7/-1/
```

Con toda la información contenida en este archivo INFO.txt el desarrollador en este momento de la ejecución tiene que crear su archivos de configuración antes de poder pasar a la parte de instalación de la aplicación.

4.4.2 **INSTALACIÓN**

Al igual que para la parte del despliegue, se irá explicando la función que realizaba la primera versión y seguidamente, los cambios de criterio y código para la versión 2.0 en la columna de la derecha.

En este punto, como se comentó anteriormente se debe contar de partida con: el archivo XML, los archivos de configuración definidos correctamente por el desarrollador y los archivos que lanzarán los experimentos. Hay que recordar que para estas pruebas estos últimos archivos no son necesario debido a que se parte de la aplicación inicial y por tanto se tiene la herramienta necesaria para ir creándolos dinámicamente, pero en cualquier otro caso, el desarrollador debe poner el nombre de dichos archivos, así como su path en el lugar que corresponde dentro del XML.

El despliegue de la red terminaba su ejecución con la creación del archivo con todos los datos del experimento; por tanto, se debe empezar la instalación leyendo dicho archivo y extrayendo su información para poder ser usada durante la configuración y ejecución de los experimentos, lo cual muestra una gran diferencia con la versión 1.0 porque todo este proceso se hacia automáticamente a la hora de ejecutar la herramienta ExperimentoP2PP.

CreaExperimento 1.0 (732 líneas)	**CreaArchivos** 2.0 (500 líneas)
1. En una sola ejecución de esta clase se creaban todos los archivos necesarios para las probabilidades marcadas (1/k, 0´3, 0´6 y 0´9) y para los dominios establecidos (1, 5, 10 y 20)	**1.** Debido a la introducción del XML para poder configurar con más detalle los experimentos, en una ejecución sólo se crean los archivos necesarios para un determinado número de dominios y para las probabilidades anteriores.

Hay que comentar también, que aunque la versión 2.0 tiene como objetivo generalizar una aplicación ya creada como se ha dicho en varias ocasiones, no deja de ser una particularización, debido a que para este caso se ha el software creado en 1.0 pero con los parámetros del caso particular porque esta es la única forma de poder validar el diseño y desarrollo de la aplicación obteniendo resultados similares a los de la versión de partida. Por tanto, para validar este proyecto se usarán algunas clases de la versión 1.0 que en cualquier otro caso no hay que usar y se sustituye únicamente depositando los archivos lanza4experimentoedgeX.sh en una carpeta e indicándolo en el XML en su lugar correspondiente.

Una vez comentadas estas salvedades, se va a proceder a explicar todo el proceso que se realiza durante la instalación. Hay que tener siempre en cuenta que todo lo que en la versión 1.0 se establecía directamente, en esta se ha automatizado por medio del XML principal.

CreaExperimento 1.0 (732 líneas)	**CreaArchivos** 2.0 (500 líneas)
2. Se le asigna a cada peer su función y su dominio dentro del experimento.	**2.** Este proceso como se ha comentado antes se realiza en la parte de despliegue de la red en la clase Ejecución porque es necesario saber esos datos para elaborar el archivo INFO.txt y así el desarrollador cree adecuadamente sus archivos de configuración.
3. Crea los 4 archivos necesarios para la ejecución (lanza1bootedgeX.sh, lanza2inicioedgeX.sh, lanza3superedgeX.sh y lanza4experimentoedgeX.sh).	**3.** Realiza la misma función pero todo el proceso está automatizado en base a los datos introducidos en el XML principal.

RecogeDatos (igual en las dos versiones)

Son iguales en ambas versiones porque los archivos a recoger una vez finalizados todos los experimentos son los mismos.

AnalizaResultados (igual en las dos versiones)

Son iguales debido a que el formato de los datos recogidos en ambas versiones son iguales; por tanto, se analizan de la misma manera.

4.5 Implementación de mi aplicación

En este apartado se va a explicar la principal modificación realizada respecto a la versión de partida, pero sin entrar en mucho nivel de detalle debido a que en los apartados anteriores se explicó la primera versión. Los grandes cambios que se han realizado principalmente son, la introducción del XML que automatiza todo el proceso y en la creación de la clase Ejecución durante el despliegue de la red, que es la que se va a explicar en este apartado dejando la explicación de cómo se usa y que significa cada etiqueta del XML para el siguiente apartado de este capítulo.

La clase Ejecución es la última que se usa para el despliegue de la red, justamente después de que se hayan creado los archivos .route y .model y además todas las máquinas que participarán en el experimento contengan dichos archivos. Su función principal es la de mostrar al desarrollador el escenario creado en el que realizaremos posteriormente los experimentos.

Partiendo del archivo con el modelo que ha creado ModelNet (.model), se va a ir procesando, almacenando todos los datos (numero de virtual node, máquina virtual donde se encuentra, etc.) en un Vector. Se van leyendo las etiquetas que definen cada dominio en el XML para ir asignando las funciones a los peers en el orden que están en el XML.

En primer lugar, se lee la etiqueta del número de checkers que habrá en el experimento y luego se obtiene el número final de peers participantes, después de restar todos los superpeers, bootstraps y superbootstrap. Se van leyendo todas las etiquetas correspondientes a los dominios que aparecen en el XML, para finalmente mostrar toda la información de los arrays de Nodos Virtuales en el archivo INFO.txt.

Comentar brevemente que en la parte de instalación también se han realizado modificaciones para poder partir desde los archivos que lanzan los experimentos pero que como anteriormente hemos comentado, esta opción no se va a usar y se crearán dinámicamente dichos archivos, tal y como se hacía en la primera versión, pero destacar por tanto que la funcionalidad está creada y sólo necesita unas pequeñas modificaciones para activarla.

4.6 **Formato del archivo XML**

Aunque este apartado sea uno de los últimos dentro del capítulo, hay que decir que es uno de los más importantes porque en él se va a explicar el archivo desde el que parte toda la aplicación desarrollada y por tanto, si está mal configurado, nada de lo que se ha explicado en las secciones anteriores funcionará correctamente. Es importante que el desarrollador preste mucha atención y tenga muy claro el funcionamiento de la aplicación para así poder ubicar correctamente todos los archivos y aplicaciones dentro de las máquinas virtuales y físicas.

En el apartado 4.2 se expuso un ejemplo de archivo XML para la configuración de un experimento con un dominio, pues bien; ahora se va a ir explicando cada una de las cinco partes de las que consta el XML detalladamente, debido a que es fundamental la comprensión de este archivo para la correcta configuración de los experimentos.

4.6.1 **Sección de configuración de red**

En primer lugar se indica la versión de XML y el lugar donde se aloja el fichero DTD, que se mostrará en el siguiente apartado y que contiene las pautas para una correcta formación del archivo XML. La etiqueta general de esta parte se llama *red*, y las de su interior son:

- *machines*: Nombre y directorio donde se encuentra el fichero con las máquinas participantes que usará ModelNet (.hosts).
- *topologia*: Nombre y directorio donde se encuentra el fichero con la topología central de red que usará ModelNet (.graph).
- *salida*: Nombre que se asignará al fichero con la topología general, al fichero con el modelo de la red (.model) y al fichero con las rutas de ModelNet (.route). Además se nombrará al archivo ejecutaModelNet*salida*.sh para que se reconozca mejor de que experimento se trata.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE proyecto SYSTEM "/home/fgomez/PFC2/ficheros/projectP2PP.dtd">
<proyecto>
  <red>
    <machines ruta = "/home/fgomez/PFC2/ficheros/"> sample.hosts </machines>          <!-- Archivo .hosts-->
    <topologia ruta = "/home/fgomez/PFC2/ficheros/"> global.graph </topologia>        <!-- Archivo .graph-->
    <salida> DESPLEGADO1 </salida>                                  <!-- Nombre de salida de los archivos creados -->
    <domain> 1 </domain>
    <num_experimentos>10</num_experimentos>
    <num_peers > 1000 </num_peers>                                  <!--Número total experimento-->
    <num_checkers> 8 </num_checkers>
    <prob> 1 </prob>
    <tiempo_total>3000</tiempo_total>
    <archivo_general ruta = "./archivos/experimentos/"/>             <!-- Path donde se crearan arch de experimento -->
    <nom_domain> .mnuc3m.es </nom_domain>
    <control nombre = "control">163.117.140.36</control>
  </red>
```

- *domain*: Número de dominios que habrá en el experimento.
- *num_experimentos*: Repeticiones que se realizarán de un mismo escenario.
- *num_peers*: Número de peers que participarán en el experimento.
- *num_checkers*: Número de checkers que participarán en el experimento.
- *prob*: Inicialmente será 1 porque internamente el programa realiza los experimentos con las probabilidades 1/k, 0´3, 0´6 y 0´9.
- *tiempo_total*: Tiempo que durará el experimento (sin contar el tiempo necesario para desplegar la red).
- *archivo_general*: Directorio general donde se van a crear todas las carpetas correspondientes a cada probabilidad para poder ejecutar un experimento completo.
- *nom_domain*: Nombre del dominio privado que tienen las máquinas participantes.
- *control*: Nombre y dirección IP de la máquina que hará la función de control.

Esta primera parte no tiene mucho nivel de abstracción porque todos los datos que contienen son muy generales, pero en las siguientes partes hay que tener especial atención y cuidado para generar bien los directorios correctos en las máquinas que corresponda.

4.6.2 **Sección de configuración de dominios**

Esta segunda parte es la que se encarga de la configuración de cada dominio, estableciendo cuantos peers de cada tipo va a tener y como se va a ejecutar cada uno de ellos.

Todas las etiquetas que se van a explicar a continuación están dentro de la principal que se llama *dominios*.

- *checkers*: Número de checkers que habrá en el experimento. Comprobar que este número corresponde con el indicado en la primera parte en la etiqueta *num_checkers.*
- *numero*: Define completamente a un dominio indicando el número de dominio que es, cuantos peers habrá en él y cómo se distribuirán dichos peers. Se irán detallando uno a uno los tipos de peers que hay, el número y cómo se ejecutan con las siguientes etiquetas:

```
<dominios>
    <checkers n_peers = "8"/>
    <numero id = "1" n_peers = "1000" distribucion = "random">
      <node type = "0">                                    <!-- BOOTSTRAP y BOOTSTRAPSUPERPEERS-->
        <n_peers>2</n_peers>
        <opciones ejecucion = "$ruta1$ruta2$ruta3$ruta4$">                    <!-PARTE DE INSTALACIÓN->
          <ruta1> /root/p2pp/p2pmain </ruta1>                                 <!-PARTE DE INSTALACIÓN->
          <ruta2> /root/p2pp/configboot.txt </ruta2>                          <!-PARTE DE INSTALACIÓN->
          <ruta3> /root/p2pp/boot$node.log</ruta3>                            <!-PARTE DE INSTALACIÓN->
          <ruta4> nodo$node </ruta4>                                          <!-PARTE DE INSTALACIÓN->
        </opciones>
      </node>
      <node type = "1">                                    <!-- SUPERPEERS -->
        <n_peers>1</n_peers>
        <opciones ejecucion = "$ruta1$ruta2$ruta3$ruta4$">                    <!-PARTE DE INSTALACIÓN->
          <ruta1> /root/p2pp/p2pmain </ruta1>                                 <!-PARTE DE INSTALACIÓN->
          <ruta2> /root/p2pp/configsuper1.txt </ruta2>                        <!-PARTE DE INSTALACIÓN->
          <ruta3> /root/p2pp/super$node.log</ruta3>                           <!-PARTE DE INSTALACIÓN->
          <ruta4> nodo$node@dominio1</ruta4>                                  <!-PARTE DE INSTALACIÓN->
        </opciones>
      </node>
    </numero>
  </dominios>
```

- *node*: Indica el tipo de peer y contiene etiquetas de como ejecutarlos:
  - · 0 = Bootstraps y bootstrapsuperpeers.
  - · 1 = Superpeers.
  - · 2 = Peers.
  - · 3 = Checkers.
  - *n_peers*: Número de peers de este tipo en el dominio.
  - *opciones*: Contiene todos los comandos necesarios para ejecutar un determinado peer.

En el caso de los checkers (tipo 3), habrá que especificar cómo se llama el archivo que configurará ese tipo de peers y en que directorio se encuentra.

4.6.3 **Sección de configuración de directorios**

Esta sección del XML es muy sensible a fallos porque en ella se encuentran todos los directorios que se van a utilizar en el proceso de creación, despliegue, instalación y ejecución de un experimento, y por tanto, cualquier error en la escritura de uno de ellos puede dar lugar a que un archivo de configuración no se encuentre en el directorio correcto cuando la aplicación lo requiera y por tanto la ejecución no funcione correctamente.

Todas las etiquetas de esta sección están dentro de la principal que se llama *instalar*. Se llamará máquina principal a aquella donde se ha desarrollado el software. Las etiquetas son:

- *inicio_mi_ruta*: Directorio en la máquina principal que contendrá los archivos creados en la parte de despliegue (despliegaModelNet.sh, ejecuta1.sh y ejecutaModelnet___.sh), el archivo .model que se crea durante el despliegue de la red para que sea usado en la instalación.

```
<instalar>
   <inicio_mi_ruta ruta= "/home/fgomez/PFC2/MiPFC/archivos/instalar/"/>
   <inicio_mi_ruta_INFO
ruta="/home/fgomez/PFC2/MiPFC/archivos/instalar/INFO/">INFORMACION1.txt</inicio_mi_ruta_INFO>
   <inicio_mi_ruta_admin ruta = "/home/fgomez/PFC2/MiPFC/archivos/administracion/"/>
   <inicio_mi_ruta_config ruta = "./archivos/configuracion/"/>                    <!-PARTE DE INSTALACIÓN->
   <inicio_mi_ruta_app ruta = "./archivos/aplicaciones/"/>
   <inicio_mi_ruta_resultados ruta = "/home/fgomez/PFC2/MiPFC2/archivos/resultados/"/>

   <inicio_dir_local ruta = "/home/control/Aproject/instalar/"/>
   <inicio_dir_local_admin ruta = "/home/control/Aproject/"/>
   <inicio_dir_local_zip ruta = "/home/control/Aproject/experimentos"/>
   <inicio_dir_local_ejecucion ruta = "/home/control/Aproject/archivos/experimentos/"/>
   <inicio_dir_local_config ruta = "/home/control/Aproject/configuracion/"/>      <!-PARTE DE INSTALACIÓN->
   <inicio_dir_local_app ruta = "/home/control/Aproject/aplicaciones/"/>

   <inicio_dir_remota ruta = "/usr/modelos/"/>
   <inicio_dir_remota_ejecucion ruta = "/root/p2pp/"/>
   <inicio_dir_remota_raiz ruta = "/root/"/>
   <inicio_file_zip ruta = "./archivos/experimentos"/>
</instalar>
```

- *inicio_mi_ruta_INFO:* Nombre y directorio en la máquina principal donde se almacenará el archivo con toda la información de los peers y clusters del experimento.

- *inicio_mi_ruta_admin:* Directorio en la máquina principal donde se almacenarán los archivos necesarios para realizar funciones de administración como reiniciar las interfaces de red.

- *inicio_mi_ruta_config:* Directorio en la máquina principal donde el desarrollador tiene que depositar los ficheros de configuración de todos los tipos de peers.

- *inicio_mi_ruta_app:* Directorio en la máquina principal donde se almacenarán las aplicaciones que se van a instalar y ejecutar.

- *inicio_mi_ruta_resultados:* Directorio en la máquina principal donde se almacenarán todos los resultados de los experimentos.

- *inicio_dir_local:* Directorio en la máquina de control donde se almacenarán los archivos necesarios en el proceso de instalación.

- *inicio_dir_local_admin:* Directorio en la máquina de control donde se almacenarán los archivos necesarios para la administración de la red.

- *inicio_dir_local_zip:* Directorio en la máquina de control donde se almacenará el archivo final comprimido que crea la aplicación con todos los experimentos.

- *inicio_dir_local_ejecucion:* Directorio en máquina de control donde descomprimirán todos los archivos necesarios para ejecutar los experimentos.

- *inicio_dir_local_config:* Directorio en máquina de control donde se almacenarán los archivos de configuración de los diferentes tipos de peers.

- *inicio_dir_local_app:* Directorio en máquina de control donde se almacenarán las aplicaciones que se van a utilizar durante los experimentos.

- *inicio_dir_remota:* Directorio principal en edges/cores donde se almacenarán los archivos necesarios para la instalación.

- *inicio_dir_remota_ejecucion:*Directorio en edges/cores donde se almacenarán los archivos necesarios para la ejecución de los experimentos.

- *inicio_dir_remota_raiz:* Directorio en edges/cores principal.

- *inicio_file_zip:* Directorio en máquina principal donde se va crear el archivo comprimido con todos los experimentos.

Esta parte es importante para que los experimentos se lleven a cabo correctamente, porque un error al introducir un nombre de directorio, puede por ejemplo, propiciar que varios archivos requeridos por el sistema para ejecutar los experimentos no sean encontrados y por tanto dar error.

4.6.4 **Sección de configuración de aplicaciones**

Esta parte del XML contiene los nombres y directorios en donde se encuentran las aplicaciones que van a ser instaladas y ejecutadas. En este caso se han utilizado las mismas que en la primera versión; es decir, p2pmain y checker, que puede verse su funcionamiento en [SENB07a]. La etiqueta que contiene los datos es *aplicaciones*.

```
<aplicaciones numero = "2">
   <app ruta = "./archivos/aplicaciones/" > p2pmain </app>
   <app ruta = "./archivos/aplicaciones/" > checker </app>
</aplicaciones>
```

4.6.5 **Sección de configuración de experimentos**

La última parte es, en un caso general, la más importante y compleja para el desarrollador, debido a que hay que almacenar y poner el nombre de los archivos que contienen las instrucciones a ejecutar durante los experimentos. Para este caso, como se viene diciendo en todo el capítulo, se ha usado el programa que genera dichos archivos, pero cada desarrollador deberá contar con una aplicación que genere estos archivos y luego indicar donde los deja para que puedan ser usados por la aplicación. Por tanto, si durante la ejecución de un experimento hay un error, será objetivo de cada desarrollador comprobar sus archivos lanza4experimentoedgeX.sh.

```
   <experimentos numero = "2">
 <edge1 ruta = "/home/fgomez/MiPFC2/archivos/configuracion/">lanza4experimentoproxmoxedge1.sh</edge1>
 <edge2 ruta = "/home/fgomez/MiPFC2/archivos/configuracion/">lanza4experimentoproxmoxedge2.sh</edge2>
   </experimentos>
</proyecto>
```

4.7 **DTD**

Este archivo también tiene mucha importancia dentro del proyecto debido a que el XML es la pieza fundamental. El DTD que se muestra a continuación será el encargado de validar dicho XML, siendo este la base en la ejecución de la aplicación.

```
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT proyecto (aplicaciones|instalar|dominios|red)*>
<!ELEMENT red (control|nom_domain|archivo_general|
tiempo_total|prob|num_peers|domain|salida|topologia|machines)*>
<!ELEMENT machines (#PCDATA)>
<!ATTLIST machines
  ruta CDATA #IMPLIED
 >
<!ELEMENT topologia (#PCDATA)>
<!ATTLIST topologia
ruta CDATA #IMPLIED
 >
<!ELEMENT salida (#PCDATA)>
<!ELEMENT domain (#PCDATA)>
<!ELEMENT num_peers (#PCDATA)>
<!ELEMENT prob (#PCDATA)>
<!ELEMENT tiempo_total (#PCDATA)>
<!ELEMENT archivo_general EMPTY>
<!ATTLIST archivo_general
 ruta CDATA #IMPLIED
 >
<!ELEMENT nom_domain (#PCDATA)>
<!ELEMENT control (#PCDATA)>
<!ATTLIST control
   nombre CDATA #IMPLIED
 >
<!ELEMENT dominios (numero|checkers)*>
<!ELEMENT checkers EMPTY>
<!ATTLIST checkers
   n_peers CDATA #IMPLIED
 >
<!ELEMENT numero (node)*>
<!ATTLIST numero
   distribucion CDATA #IMPLIED
   n_peers CDATA #IMPLIED
   id CDATA #IMPLIED
 >
<!ELEMENT node (opciones|arch_log|arch_conf|n_peers)*>
<!ATTLIST node
   type CDATA #IMPLIED
 >
<!ELEMENT n_peers (#PCDATA)>
<!ELEMENT arch_conf (#PCDATA)>
<!ATTLIST arch_conf
   ruta2 CDATA #IMPLIED
   ruta CDATA #IMPLIED
 >
<!ELEMENT arch_log (#PCDATA)>
<!ATTLIST arch_log
   ruta2 CDATA #IMPLIED
 >
<!ELEMENT opciones (#PCDATA)>
<!ELEMENT instalar (inicio_file_zip|inicio_mi_ruta|
inicio_mi_ruta_INFO|inicio_mi_ruta_admin|
inicio_mi_ruta_config|inicio_mi_ruta_app|inicio_dir_local|
inicio_dir_local_admin|inicio_dir_local_zip|
inicio_dir_local_ejecucion|inicio_dir_local_config|
inicio_dir_local_app|inicio_dir_remota|
inicio_dir_remota_ejecucion)*>
<!ELEMENT inicio_mi_ruta EMPTY>
<!ATTLIST inicio_mi_ruta
   ruta CDATA #IMPLIED
          >
<!ELEMENT inicio_mi_ruta_INFO (#PCDATA)>
<!ATTLIST inicio_mi_ruta_INFO
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_mi_ruta_admin EMPTY>
<!ATTLIST inicio_mi_ruta_admin
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_mi_ruta_config EMPTY>
<!ATTLIST inicio_mi_ruta_config
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_mi_ruta_app EMPTY>
<!ATTLIST inicio_mi_ruta_app
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_local EMPTY>
<!ATTLIST inicio_dir_local
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_local_admin EMPTY>
<!ATTLIST inicio_dir_local_admin
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_local_zip EMPTY>
<!ATTLIST inicio_dir_local_zip
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_local_ejecucion EMPTY>
<!ATTLIST inicio_dir_local_ejecucion
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_local_config EMPTY>
<!ATTLIST inicio_dir_local_config
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_local_app EMPTY>
<!ATTLIST inicio_dir_local_app
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_remota EMPTY>
<!ATTLIST inicio_dir_remota
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_dir_remota_ejecucion EMPTY>
<!ATTLIST inicio_dir_remota_ejecucion
   ruta CDATA #IMPLIED
 >
<!ELEMENT inicio_file_zip EMPTY>
<!ATTLIST inicio_file_zip
   ruta CDATA #IMPLIED
 >
<!ELEMENT aplicaciones (app)*>
<!ATTLIST aplicaciones
   numero CDATA #IMPLIED
 >
<!ELEMENT app (#PCDATA)>
<!ATTLIST app
   ruta CDATA #IMPLIED  >
```

4.8 **Metodología de validación**

Para finalizar este capítulo y una vez explicados todos los puntos a tener en cuenta para poder entender y utilizar la aplicación que en este documento se está explicando, como ayuda adicional se van a indicar algunas pautas que serán muy útiles a la hora de comprobar si se está realizando todo el proceso correctamente y así no perder tiempo en el despliegue de la red y la instalación, cumpliendo por tanto con la función principal de este proyecto que no es otra que la de proporcionar al futuro desarrollador una herramienta con la que no se tenga que preocupar de nada que no sea exclusivamente la generación de su experimento particular y de mantener activas las máquinas que usará, ya que todo el proceso correrá a cargo de la aplicación automáticamente.

Para que las pautas sean más fáciles de comprender, se van a dividir en las dos partes que componen la aplicación. Con el apartado anterior en el que se explica detalladamente cada etiqueta del XML y los anteriores apartados en los que se muestran todos los detalles a tener en cuenta a la hora de iniciar una ejecución, será suficiente para solucionar la gran mayoría de fallos o inconvenientes que surjan al ejecutar la aplicación.

4.8.1 **Despliegue de la red**

Los lugares y ficheros a comprobar no son muchos porque normalmente los fallos siempre son los mismos y vienen a la hora de definir algún directorio o nombre de archivo. No obstante, se debe comprobar en primer lugar si la información que aparece a la hora de desplegar la red coincide con lo que se ha introducido en la etiqueta *red* del XML (nombre y fichero de los archivos .hosts y .graph, cuantas máquinas hay en el experimento, etc.).

La máquina que actúa como control ya se ha comentado que es muy importante dentro del organigrama de ejecución de ambas partes y en esta, lo que hay que comprobar es tener:

- En el directorio principal, el script reinciaRed.sh.
- En el directorio aplicaciones, las aplicaciones que se instalarán y ejecutarán.
- En el directorio configuración ningún archivo.
- En el directorio instalar, los archivos graph, hosts, model, route y despliegaModelnet.sh.
- El archivo ejecuataModelNetDESPLEGADOX.sh correctamente creado.

En las máquinas virtuales que actúan como cores y edges hay que comprobar que se tiene:

- En el directorio /usr/modelos, los archivos model y route.
- En el primer core del archivo hosts también los archivos hosts y graph.

En la máquina donde se ha desarrollado el software hay que comprobar que al finalizar la ejecución de la parte de despliegue está el archivo .model, para posteriormente poder usarlo durante la instalación.

Como se ha comentado, la gran mayoría de los problemas vienen dados por alguno de estos motivos, no obstante hay que decir que pueden existir más, pero ocurrirán con menor frecuencia o serán ajenos a la ejecución de la aplicación.

4.8.2 **Instalación**

Antes de comenzar a comentar las comprobaciones que hay que hacer durante la ejecución de esta segunda parte, hay que decir que los mayores problemas vienen dados por un fallo humano a la hora de crear los archivos de configuración con la información que se muestra en el archivo INFO.txt.

Una vez ejecutado el script que lanza los experimentos es difícil darse cuenta de algunos fallos, por eso, se recomienda que antes de hacerlo, se compruebe la conectividad entre todas las máquinas que van a formar parte del experimento, así como alguna prueba de conectividad entre algunos nodos virtuales. Después de ejecutar el script que lanza el experimento se debe comprobar a la vez que se ejecuta varios pasos importantes tales como:

- Ver *deployhost* correcto en todos los cores.
- Ver *deployhost* correcto en todos los edges mediante el archivo deployhost.txt.
- Ver en control que se van ejecutando los scripts según su orden:
    - Boots, superpeers y peers *OK*.
    - Pings para comprobar conectevidad *OK*.
    - COMPLETADO _____ en edgeX.
    - Recogida de archivos satisfactoria.

Al igual que en la primera parte, esta serie de comprobaciones son las que hay que realizar con mayor frecuencia para que la ejecución de los experimentos sea satisfactoria, pero siempre cabe la posibilidad de que existan otro tipo de problemas, normalmente ajenos a la aplicación.

Capítulo IV

VALIDACIÓN

Este capítulo será el encargado de mostrar los resultados que se han obtenido usando la herramienta creada para este proyecto, configurándola con los parámetros usados para la aplicación de partida, obteniendo así la mejor forma de comprobar que la aplicación esta bien diseñada e implementada, y por tanto, consiguiendo el ya comentado objetivo de generalizar una aplicación ya diseñada pero reduciendo la complejidad para el desarrollador a la hora de utilizarla. Por tanto, lo mostrado en este capítulo será exclusivamente la manera de validar la aplicación respecto a la de partida. También se indicará de dónde proceden los datos que se han usado para ambas aplicaciones.

En el siguiente capítulo se mostrarán los resultados obtenidos configurando la aplicación con unos parámetros totalmente nuevos, en un paso más de demostrar que se pueden realizar experimentos extremadamente complejos, cosa que no era posible con la aplicación de partida porque era necesario diseñar y retocar muchas clases Java, pero que por contra, en este caso, únicamente hay que configurar correctamente el archivo XML principal.

1 Introducción

Todos los resultados obtenidos han tenido un mismo origen. Los datos utilizados tanto para validar la aplicación creada como para la generación de nuevos experimentos son, al igual que para la aplicación de partida, los del proyecto ARK de CAIDA [CAID10] y el proyecto PingER [PIN10]. A partir de ellos, se crea la topología en la que posteriormente se pueden emular los retardos reales que sufren los paquetes al ser transmitidos entre los diferentes países.

En el proyecto CAIDA (Asociación Cooperativa para el análisis de Datos en Internet) colaboran organizaciones comerciales, gobiernos y los sectores de investigación encaminados a promover una mayor cooperación en la ingeniería y el mantenimiento de una infraestructura robusta, escalable y global de Internet.

1.1 Proyecto CAIDA

Tiene como función investigar de forma práctica y teórica varios aspectos de Internet para:

- Dar una idea de la función macroscópica de la infraestructura de Internet, así como su comportamiento, uso y evolución.

- Promover un entorno de colaboración en el que pueden adquirirse los datos, analizarlos y (según corresponda) compartirlos.

- Mejorar la integridad del campo científico de Internet.

- Informar a la ciencia, la tecnología y comunicar las políticas públicas.

Dentro de CAIDA se engloban muchos proyectos entre los que se encuentran Macroscopic Topology Project y Archipielago Measurement Infrastructure (Ark).

Del primero se puede decir que se inició en 1998 y en él se monitorizaba la conectividad global de Internet mandando paquetes de prueba desde un conjunto de monitores origen hasta cientos o miles de destinos distribuidos por todo el planeta.

Los datos recogidos de él, caracterizaban la conectividad global y el rendimiento de Internet, también permitía varias representaciones topológicas y geográficas con diferentes grados de granularidad, además de proporcionar una base empírica muy importante para modelar el comportamiento y las propiedades de Internet.

Después de una década recogiendo datos, en Febrero de 2008, este proyecto dejo paso a Ark. Este nuevo proyecto usa una evolución de las herramientas usadas por su antecesor, de modo que consigue reducir el esfuerzo destinado a desarrollar y desplegar las sofisticadas medidas a gran escala, además de proveer una infraestructura a la comunidad de medida que permite a los colaboradores realizar sus propias medidas a través de una plataforma distribuida segura.

Ark está diseñado específicamente para la medida en redes activas, lo que hace que sea más simple que otras plataformas distribuidas de propósito general, lo que permite concentrarse en proveer las utilidades necesarias para ayudar a la investigación de la red. Por tanto, ofrece un servicio de comunicación y coordinación que hace que sea más fácil desarrollar medidas distribuidas que deban trabajar juntas para lograr una meta.

1.2 **Topología utilizada**

La topología utilizada para validar la aplicación, al igual que para la aplicación de partida, se genera a partir de los datos de Ark como ya se ha comentado (caida.xml), y con la ayuda de la herramienta generada en el proyecto de partida llamada AnalizadorXML_CAIDA.

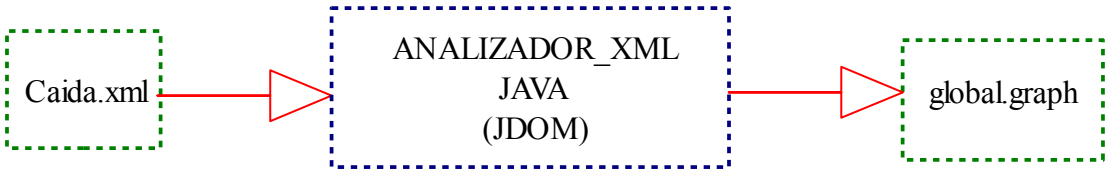


Figura 4.1: Diagrama de creación de una topología

Los países que se eligieron en su momento fueron en base al volumen de tráfico Kademlia que generaban según [SENB07c]. Los elegidos fueron, China, Francia, Italia, Dinamarca, Brasil, Alemania, Estados Unidos, Taiwan, Corea, Argentina, Portugal, Reino Unido, Japón, India, Tailandia, México, Suiza, Sudáfrica, Rusia y Australia. Para entender mejor lo explicado, se muestra a continuación un ejemplo de archivo generado usando sólo los cinco primeros países:

```
<?xml version="1.0" encoding="UTF-8"?>
<topology>
 <vertices>
  <vertex dbl_y="1" dbl_x="1" int_idx="0" role="gateway" />
  <vertex dbl_y="2" dbl_x="2" int_idx="1" role="gateway" />
  <vertex dbl_y="3" dbl_x="3" int_idx="2" role="gateway" />
  <vertex dbl_y="4" dbl_x="4" int_idx="3" role="gateway" />
  <vertex dbl_y="5" dbl_x="5" int_idx="4" role="gateway" />
 </vertices>
 <specs xmloutbug="workaround">
  <client-stub int_delayms="1" dbl_plr="0" int_qlen="100" dbl_kbps="1000" />
  <stub-transit int_delayms="1" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
  <pais1 int_delayms="1" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
  <pais2 int_delayms="5" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
  <pais3 int_delayms="4" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
  <pais4 int_delayms="5" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
  <pais5 int_delayms="17" dbl_plr="0" int_qlen="100" dbl_kbps="100000" />
 </specs>
 <edges>
  <edge specs="stub-transit" int_idx="0" int_src="5" int_dst="1" int_delayms="24" />
  <edge specs="stub-transit" int_idx="1" int_src="1" int_dst="5" int_delayms="24" />
  <edge specs="stub-transit" int_idx="2" int_src="5" int_dst="19" int_delayms="155" />
  <edge specs="stub-transit" int_idx="3" int_src="19" int_dst="5" int_delayms="155" />
  <edge specs="stub-transit" int_idx="4" int_src="5" int_dst="18" int_delayms="60" />
  <edge specs="stub-transit" int_idx="5" int_src="18" int_dst="5" int_delayms="60" />
  <edge specs="stub-transit" int_idx="6" int_src="5" int_dst="6" int_delayms="80" />
  <edge specs="stub-transit" int_idx="7" int_src="6" int_dst="5" int_delayms="80" />
  <edge specs="stub-transit" int_idx="8" int_src="5" int_dst="10" int_delayms="33" />
  <edge specs="stub-transit" int_idx="9" int_src="10" int_dst="5" int_delayms="33" />
  <edge specs="stub-transit" int_idx="10" int_src="5" int_dst="16" int_delayms="17" />
  <edge specs="stub-transit" int_idx="11" int_src="16" int_dst="5" int_delayms="17" />
  <edge specs="stub-transit" int_idx="12" int_src="5" int_dst="2" int_delayms="24" />
  <edge specs="stub-transit" int_idx="13" int_src="2" int_dst="5" int_delayms="24" />
  <edge specs="stub-transit" int_idx="14" int_src="5" int_dst="3" int_delayms="19" />
  <edge specs="stub-transit" int_idx="15" int_src="3" int_dst="5" int_delayms="19" />
  <edge specs="stub-transit" int_idx="16" int_src="5" int_dst="9" int_delayms="218" />
  <edge specs="stub-transit" int_idx="17" int_src="9" int_dst="5" int_delayms="218" />
  <edge specs="stub-transit" int_idx="18" int_src="5" int_dst="7" int_delayms="166" />
  <edge specs="stub-transit" int_idx="19" int_src="7" int_dst="5" int_delayms="166" />
  <edge specs="stub-transit" int_idx="20" int_src="5" int_dst="12" int_delayms="148" />
  <edge specs="stub-transit" int_idx="21" int_src="12" int_dst="5" int_delayms="148" />
  <edge specs="stub-transit" int_idx="22" int_src="13" int_dst="5" int_delayms="104" />
  <edge specs="stub-transit" int_idx="23" int_src="5" int_dst="13" int_delayms="104" />
  <edge specs="stub-transit" int_idx="24" int_src="13" int_dst="8" int_delayms="0" />
  <edge specs="stub-transit" int_idx="25" int_src="8" int_dst="13" int_delayms="0" />
  <edge specs="stub-transit" int_idx="26" int_src="13" int_dst="1" int_delayms="96" />
 </edges>
</topology>
```

2 **Validación de la herramienta desarrollada**

Una vez finalizado el trabajo de diseño y desarrollo de la aplicación, llega la hora de validar y comprobar que lo realizado hasta ahora es correcto y reproduce fielmente la funcionalidad de la aplicación de partida, con el comentado objetivo de tener una configuración más sencilla.

En este apartado, la dinámica que se va a seguir es la de mostrar en la parte izquierda la gráfica con los resultados obtenidos con la aplicación de partida y en la parte derecha la obtenida para este caso, pasando después a comentar lo que se estime necesario. Una vez comentado esto, se va a proceder a hacer una breve introducción de algunos detalles a tener en cuenta y seguidamente se mostrarán las comentadas gráficas.

Las gráficas en ambos casos se han realizado con Matlab. Hay que recordar que para realizar las gráficas se ha partido del archivo que genera la aplicación AnalizaResultados, de la cual ya se comentó su funcionamiento en el capítulo III, apartado 3.1.3. Al ser los experimentos iguales en ambos casos para poder validar la aplicación, se ha usado el script que se creó entonces y que va calculando para cada número de dominios y probabilidad su media e intervalo de confianza. A continuación dibuja las gráficas a partir de esos datos.

Al igual que en el caso de partida, las pruebas se han realizado con 1000 nodos, de los cuales, una media de 800 están activos durante el experimento. Se sabe que este número es ínfimo comparado con las decenas o centenas de miles que pueden participar en una red real, pero son suficientes para obtener unos resultados válidos. Comentar en último lugar que, para mejorar la precisión, cada escenario se ha repetido diez veces, obteniendo por tanto, una desviación típica muy baja e intervalos de confianza al 95% pequeños con errores menores del 10% respecto a la media estimada.

2.1 **Distribución de peers de forma determinista**

Para este caso, como ya se explicó en el capítulo III, apartado 4, la distribución de los peers será determinista; es decir, todos los clusters/países tendrán el mismo número de peers y estos se irán colocando hasta llegar al número de peers por país y seguidamente se comenzará a colocar peers en el siguiente país y así sucesivamente.

La medida que mejor definirá de aquí en adelante los resultados, y por tanto, el rendimiento de la red, es el número medio de saltos necesarios para encontrar un peer, que será precisamente la primera gráfica que se va a mostrar.

De la primera figura 4.2 se puede observar que a medida que crece el número de dominios y la probabilidad de buscar en el mismo dominio, va disminuyendo el número de saltos a realizar. En la segunda gráfica los resultados son muy similares, el porcentaje de variación de una respecto a otra es muy bajo, lo que da validez a nuestra aplicación.

En la figura 4.3 se ha cambiando en el eje de abscisas la probabilidad por el número de dominios, se puede observar mejor como tiende a disminuir el número de saltos a medida que crece el número de dominios hasta 30 dominios. Comparando ambas, se puede apreciar que los resultados no varían en exceso, únicamente el porcentaje normal que existe cuando se realizan dos experimentos con los mismos parámetros pero en distintos instantes de tiempo.

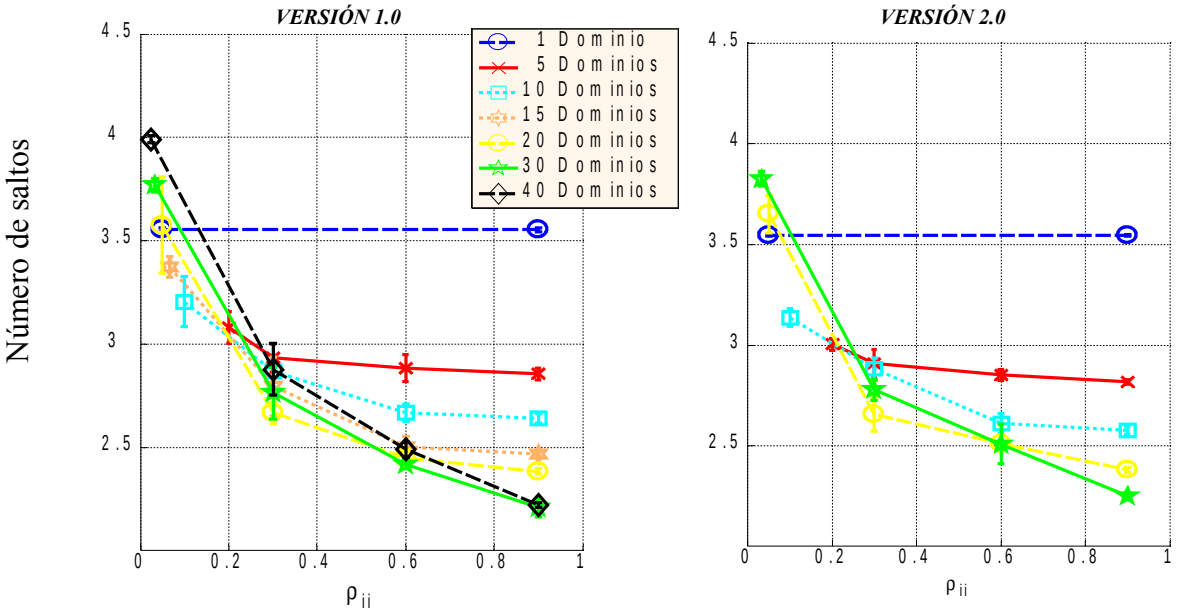


Figura 4.2: Número medio de saltos según la probabilidad

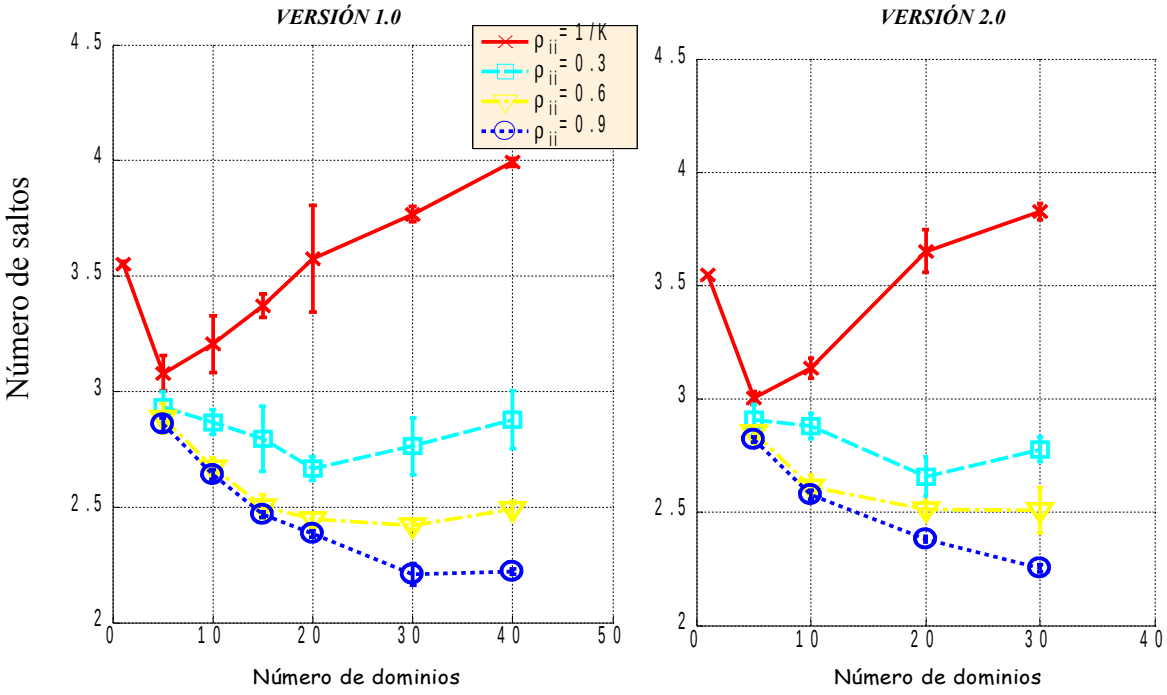


Figura 4.3: Número medio de saltos según el número de dominios

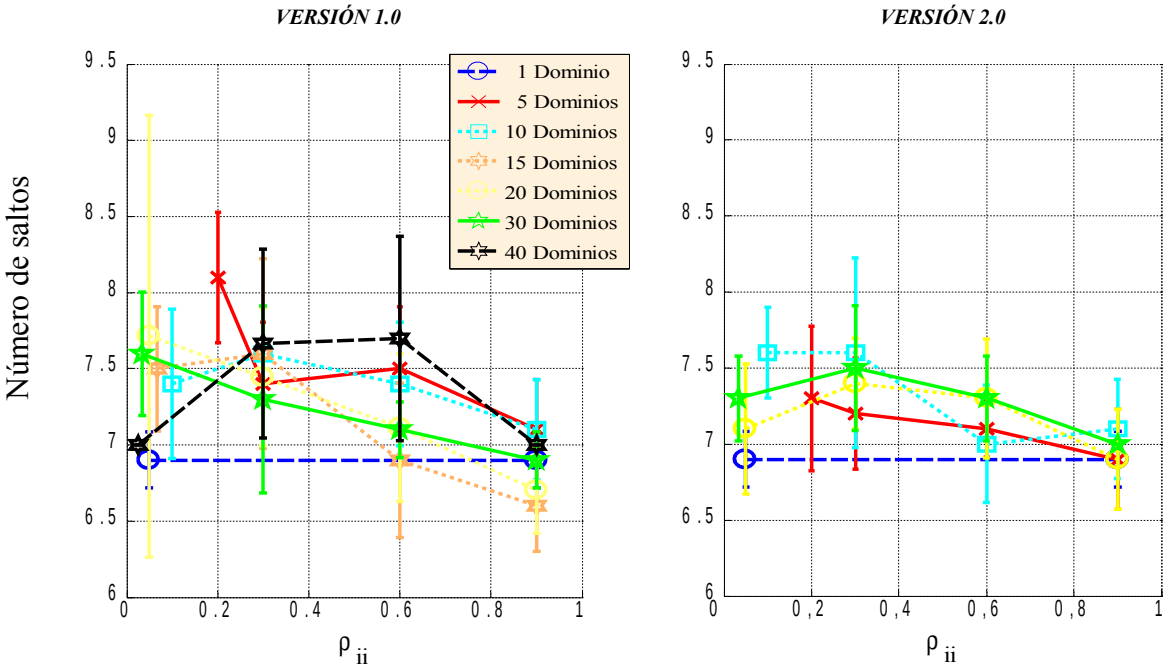


Figura 4.4: Número máximo de saltos según la probabilidad

Como puede observarse en este caso, no existe ninguna relación clara entre el número de dominios y el número máximo de saltos en la aplicación de partida, pero sin embargo, se puede ver una tendencia hacia la disminución de los saltos máximos cuando se aumenta la probabilidad de buscar en el propio dominio. Se puede comprobar que en ambos casos el número de saltos no cambia significativamente. A continuación se va a mostrar que es lo que sucede con el tiempo empleado en realizar una búsqueda.

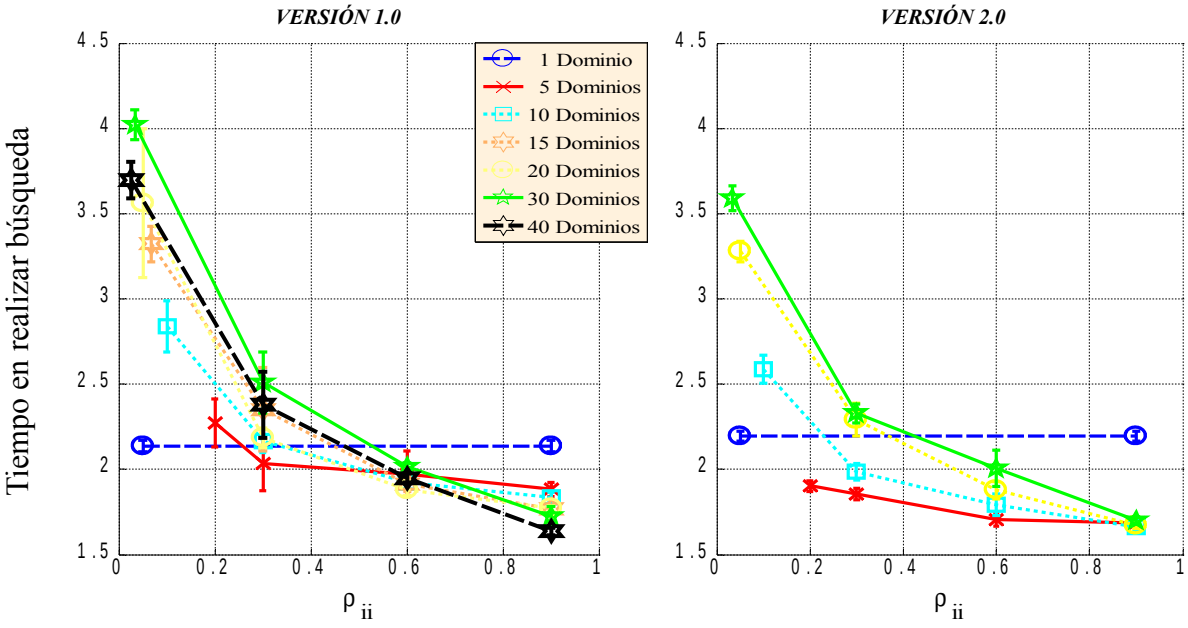


Figura 4.5: Tiempo medio en realizar búsqueda según la probabilidad

La gráfica 4.5 es interesante porque indica que el tiempo que se tarda en realizar una búsqueda para probabilidades bajas de buscar en el propio dominio es mayor para un mayor número de dominios, pero esta tendencia se invierte a medida que crece la probabilidad. Se puede suponer que la poca diferencia en los tiempos se debe a que se trabaja con un número pequeño de nodos y que cuando este número aumente, los tiempos estarán más distanciados. Contrastando ambas, se puede comprobar una vez más que las gráficas obtenidas con la aplicación son muy similares a las que se obtuvieron con la de partida.

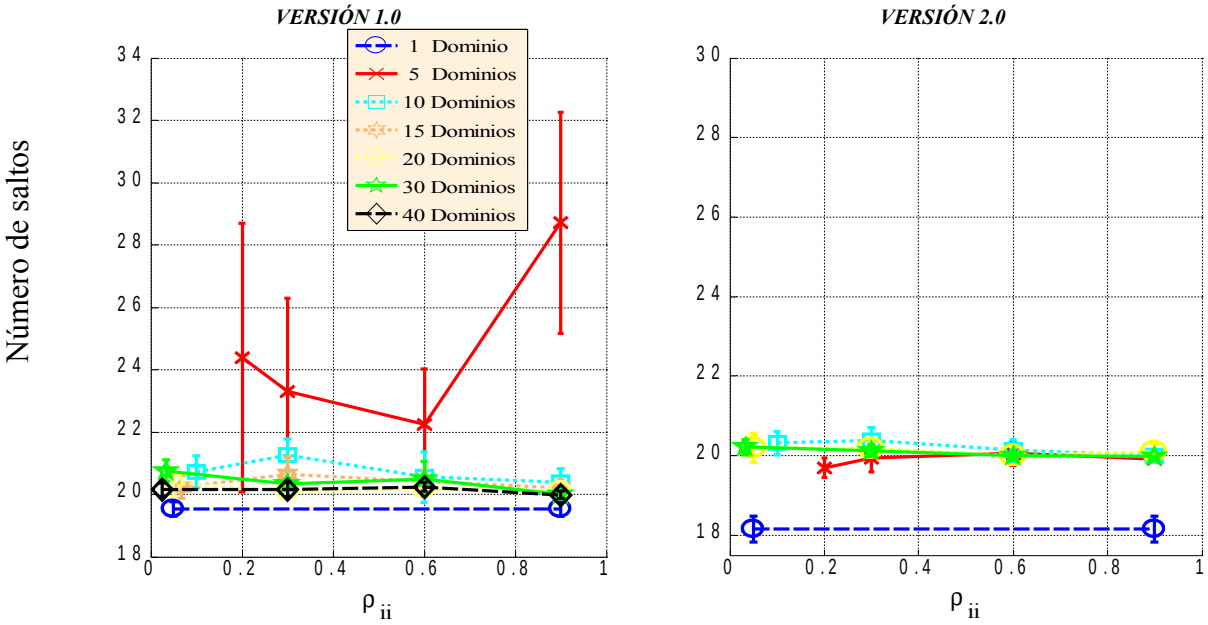


Figura 4.6: Tiempo máximo en realizar búsqueda según la probabilidad

Al igual que se hizo para el caso de los número de saltos, esta gráfica del tiempo máximo servirá como medida de la carga que sufrió el sistema durante las pruebas. Como se puede ver en ambas, los resultados son parecidos y tampoco aportan mucha información, únicamente resaltar que para un dominio el tiempo que se tarda como máximo es menor, y por tanto, la sobrecarga del equipo también lo es, lo cual es lógico porque todos los peers están en el mismo dominio y las búsquedas son siempre en dicho dominio y tardan menos en realizarse satisfactoriamente. Por otra parte, se puede ver que los valores que se obtuvieron en la aplicación de partida para 5 dominios son muy extraños y de ellos se puede decir que cuando se realizó la prueba el sistema estaría muy sobrecargado volviéndose por tanto inestable.

Hasta este punto se han contrastado los datos más relevantes en cuanto a las querys se refiere. A partir de ahora se validarán también las tablas de rutas de los nodos, para comprobar si las medidas obtenidas siguen el mismo modelo que para la aplicación de partida.

Como podía ser de esperar, en la figura 4.7 se puede ver que el número medio de entradas en la tabla de rutas de los peers va disminuyendo a medida que crece el número de nodos y comparando ambas gráficas cabe destacar que la tendencia es la misma; es decir, el número medio de entradas disminuye cuando aumenta el número de dominios. Para el caso de un dominio, el resultado es algo mayor.

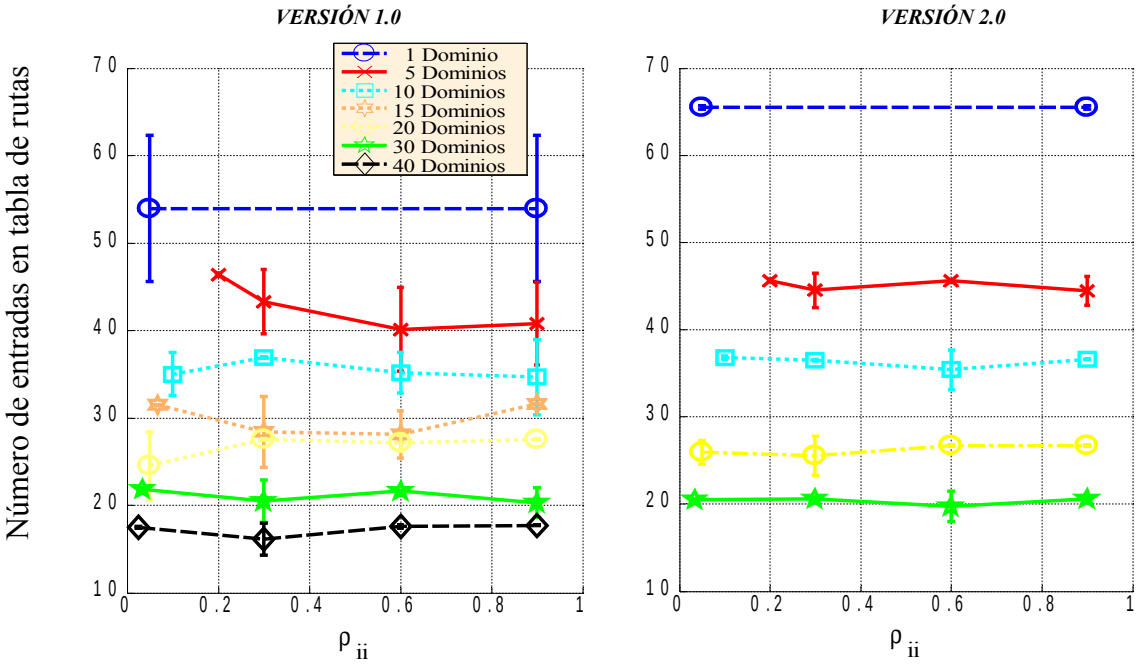


Figura 4.7: Número medio de entradas en la tabla de rutas de los peers

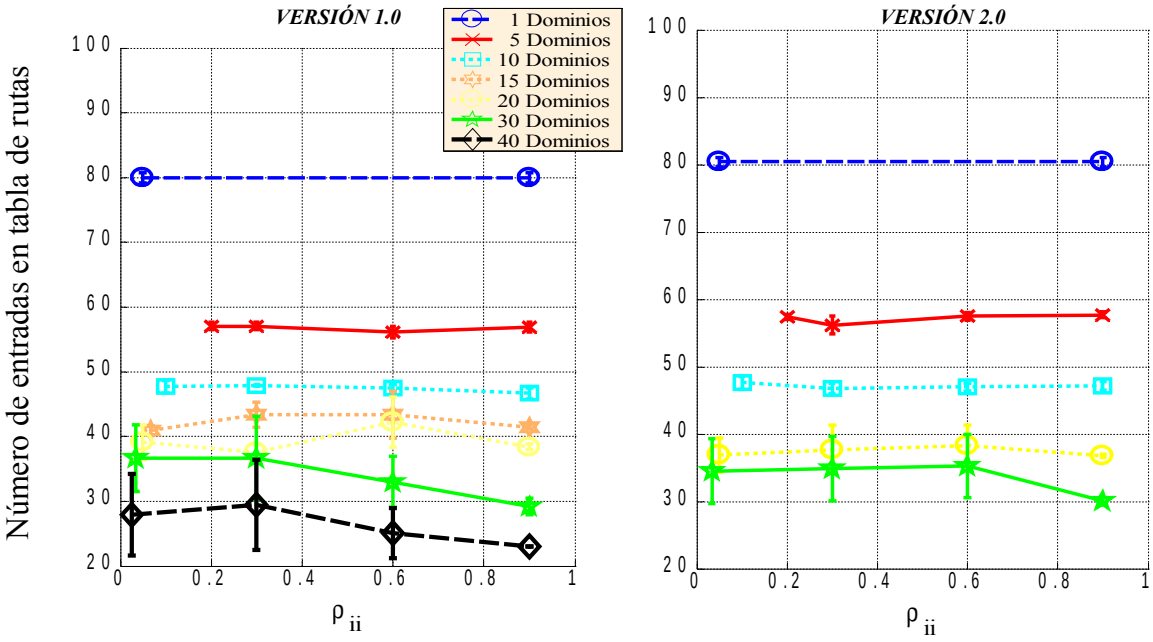


Figura 4.8: Número máximo de entradas en la tabla de rutas de los peers

Al igual que para el valor medio, los máximos van descendiendo a medida que el número de clusters aumenta. Una vez más, comparando ambas gráficas, se puede comprobar que la tendencia es la misma.

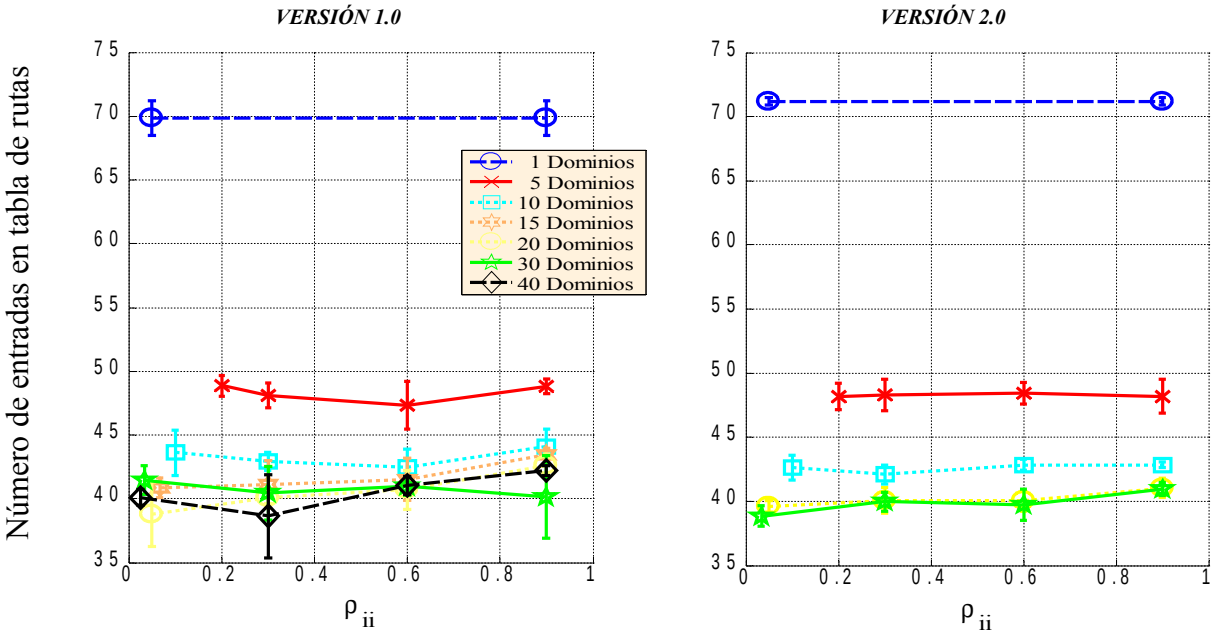


Figura 4.9: Número medio de entradas en tabla de rutas de los superpeers

Para los superpeers se puede ver que el número medio de entradas va disminuyendo a medida que aumenta el número de clusters pero que, para valores altos de dominios, como 20, 30 y 40, esta disminución no es tan pronunciada como en los otros casos. Los valores obtenidos son muy parecidos a los de la aplicación de partida, y también cumple con que para valores altos, el número de entradas no varía notablemente.

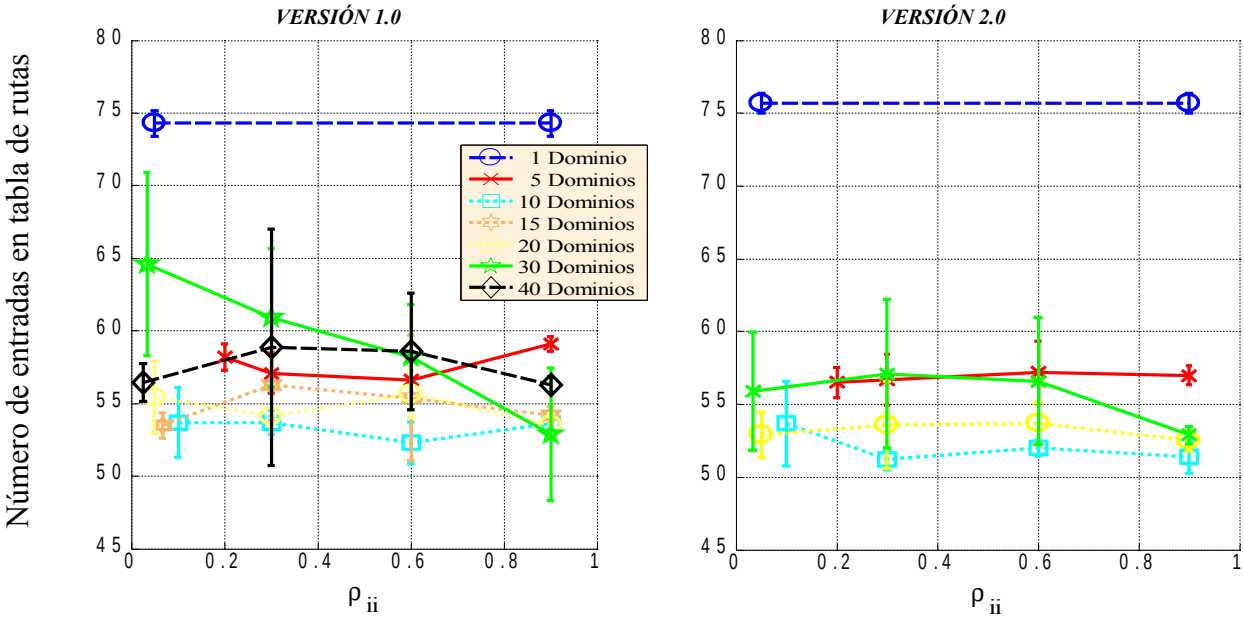


Figura 4.10: Número máximo de entradas en tabla de rutas de superpeers

En está última gráfica para el caso determinista se observa claramente como para valores de clusters altos el número de entradas es mayor, excepto para el caso de un dominio, lo que se debe a que se han manejado un número de clusters excesivo.

Para la versión 2.0 los datos obtenidos son muy parecidos y también se puede observar que para 20 dominios, el número de entradas es muy parecido al de 10, lo que también indica que puede ser un número excesivo de dominios.

2.2 **Distribución de peers de forma aleatoria**

La diferencia con el caso anterior es que ahora la colocación de los peers en cada país se realiza de forma totalmente aleatoria, dándose el caso de que peers que pertenecen al mismo dominio se encuentren en diferentes países, lo cuál puede ser un caso que se aproxime más a la realidad, pero hay que tener en cuenta que el número de peers por dominio sigue siendo el mismo.

Hay que tener en cuenta que para este tipo de escenario los resultados pueden tener mayor grado de variación con respecto a la aplicación de partida debido a que al ser todo más aleatorio, los experimentos serán mas diferentes entre sí, pero sin llegar a obtener resultados totalmente diferentes, lo cuál nos mostraría algún tipo de error en la configuración de las pruebas.

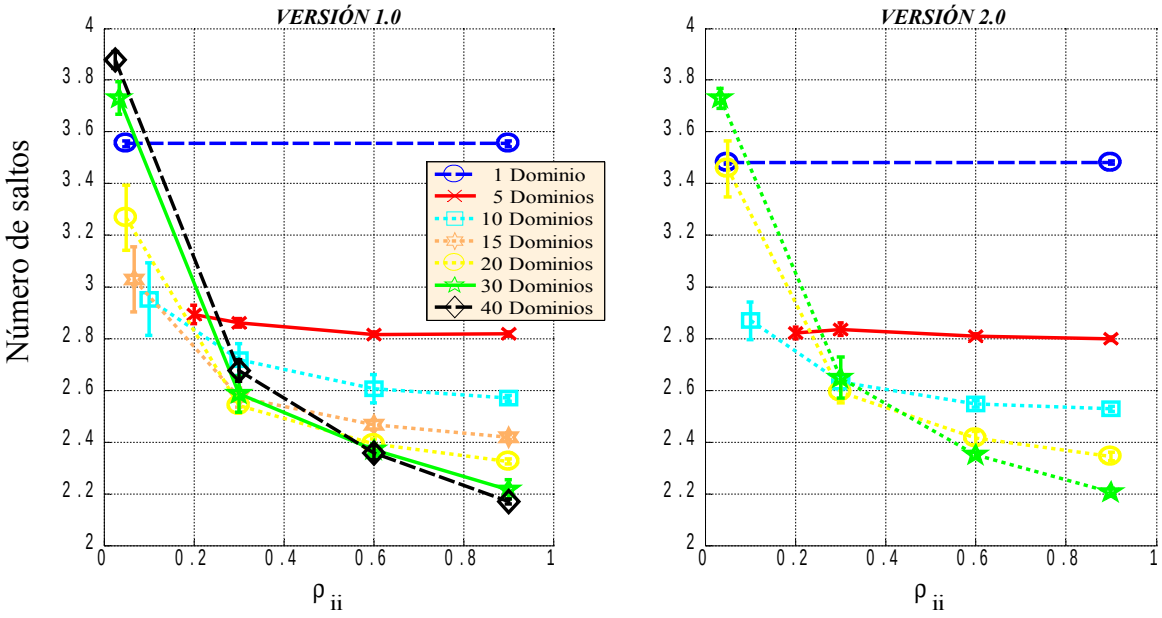


Figura 4.11: Número medio de saltos según la probabilidad

Al igual que para el caso determinista, ambas gráficas muestran la misma tendencia y valores muy parecidos.

Se puede comprobar en la figura 4.12 que los resultados obtenidos son muy parecidos, obteniendo la conclusión de que, a medida que el número de dominios aumenta (hasta 30), el número de saltos disminuye, lo que una vez más, valida la versión 2.0.

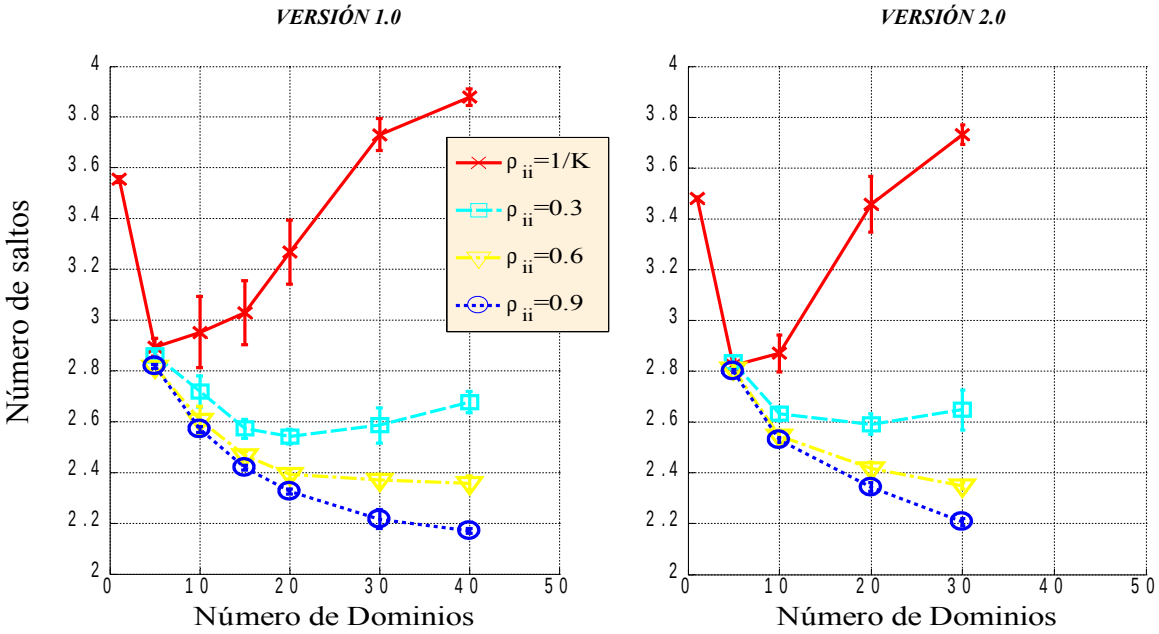


Figura 4.12: Número medio de saltos según los dominios

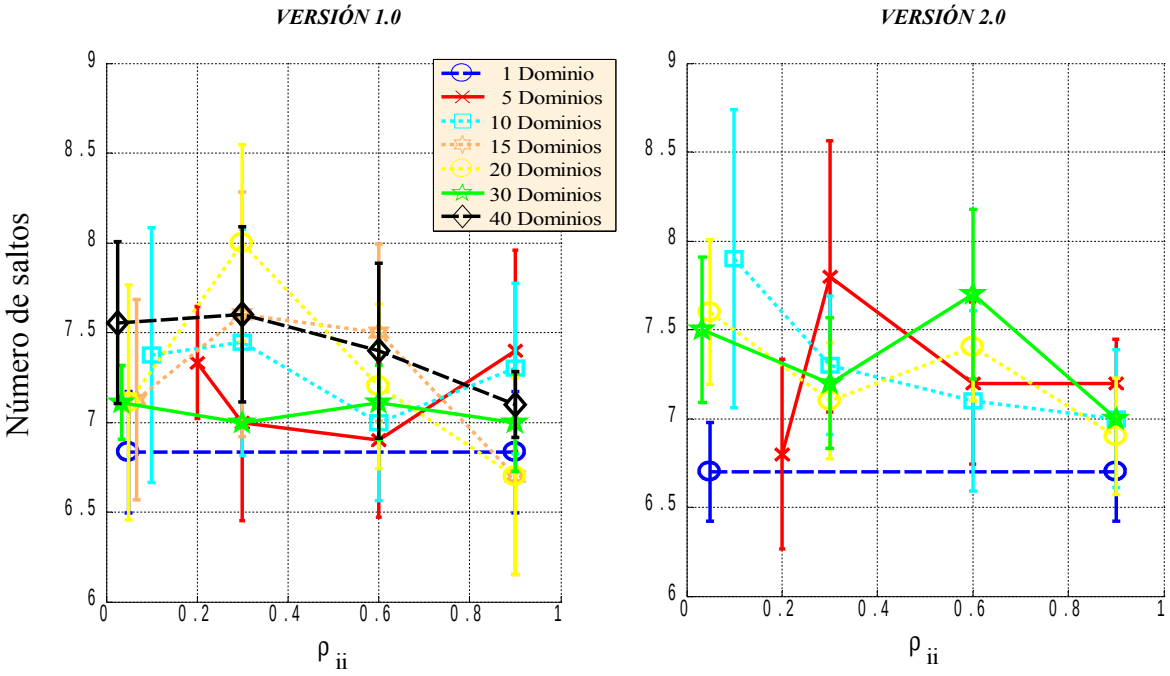


Figura 4.13: Número máximo de saltos según la probabilidad

Se puede comprobar que los resultados son muy similares y que en ambos casos no guarda ningún tipo de relación o modelo según el número de dominios.

Se observa en las gráficas de la figura 4.14 que mantienen el mismo modelo; es decir, para probabilidades bajas de buscar en nuestro propio dominio, al aumentar el número de dominios, el tiempo es mayor.

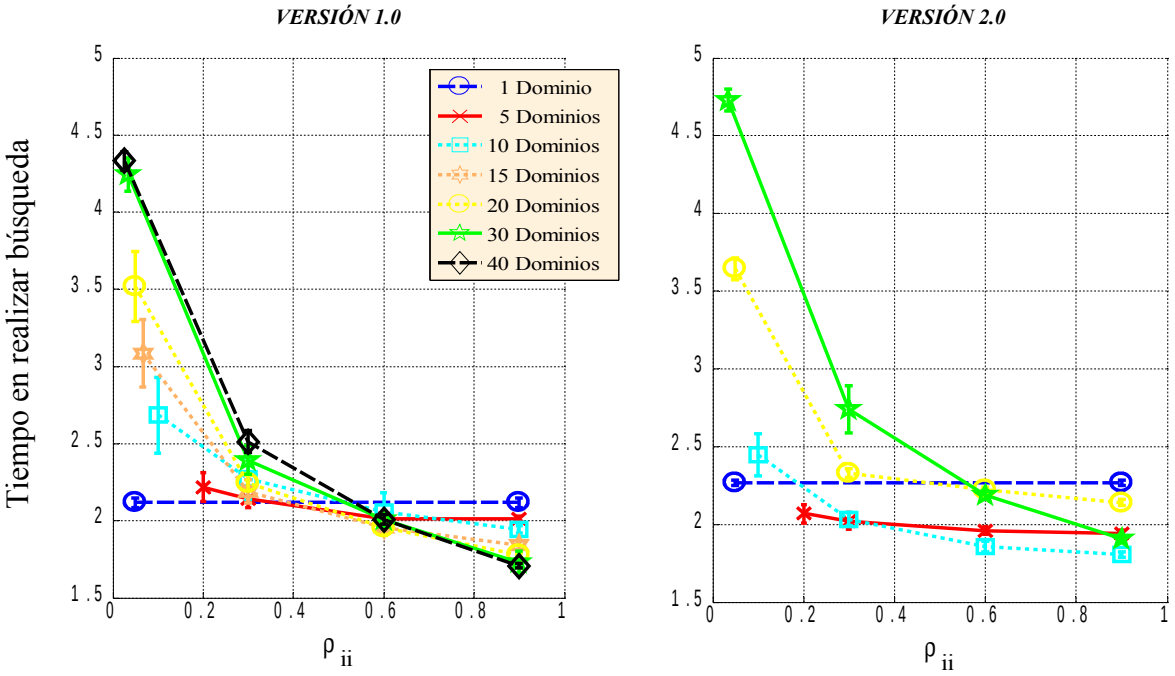


Figura 4.14: Tiempo medio en realizar búsqueda según la probabilidad

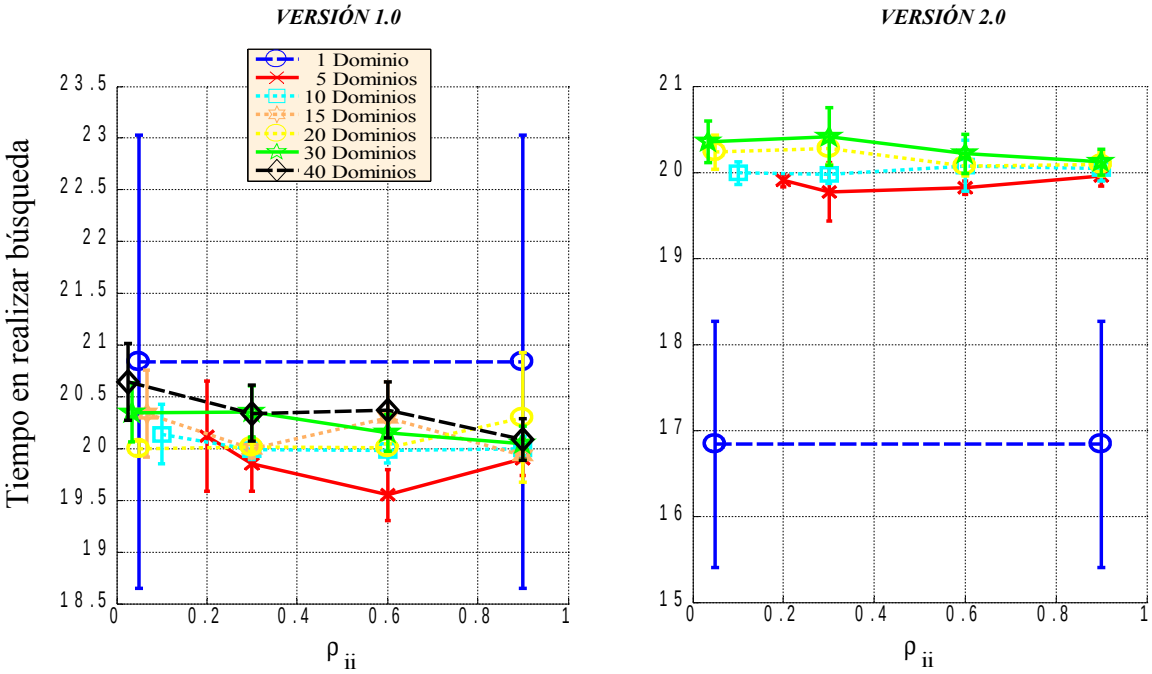


Figura 4.15: Tiempo máximo en realizar búsqueda según la probabilidad

Los resultados son similares, exceptuando para un dominio porque al ser el experimento aleatorio, depende mucho del retardo que tengamos entre un país y en otro porque para cada experimento los nodos estarán en países diferentes. Para los demás dominios son muy parecidos.

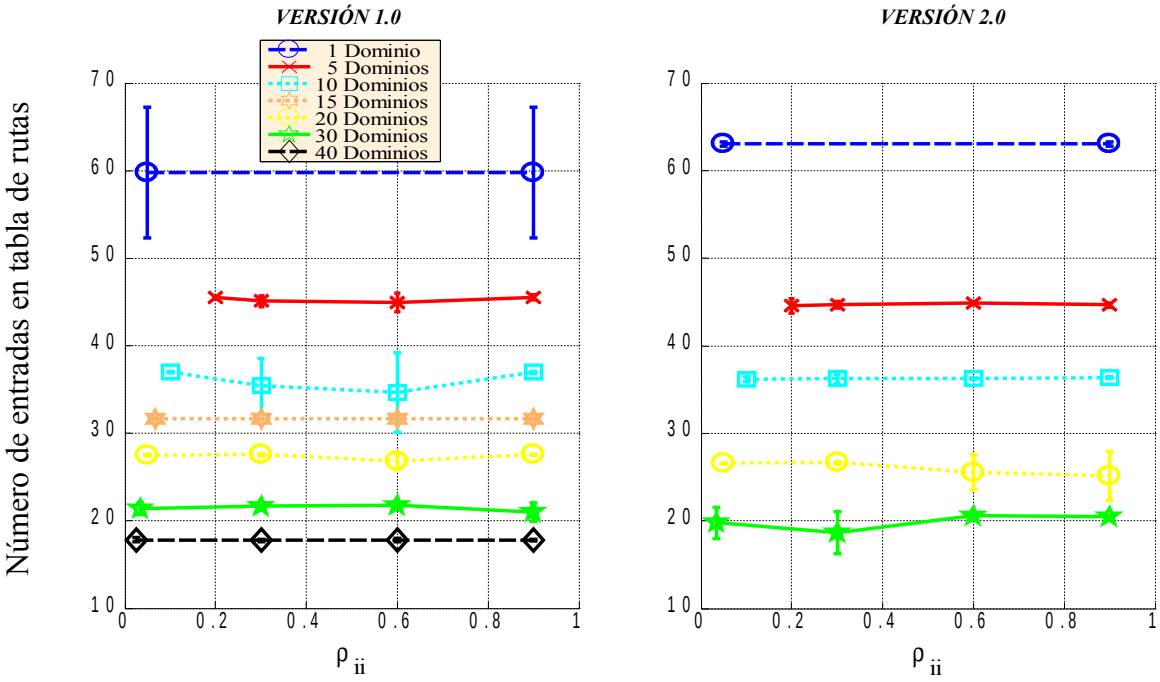


Figura 4.16: Número medio de entradas en la tabla de rutas de los peers

La tendencia que se ve es que cuantos más dominios participan, menos entradas en las tablas de rutas de los peers se realizan, y se da en ambos casos como se puede apreciar.

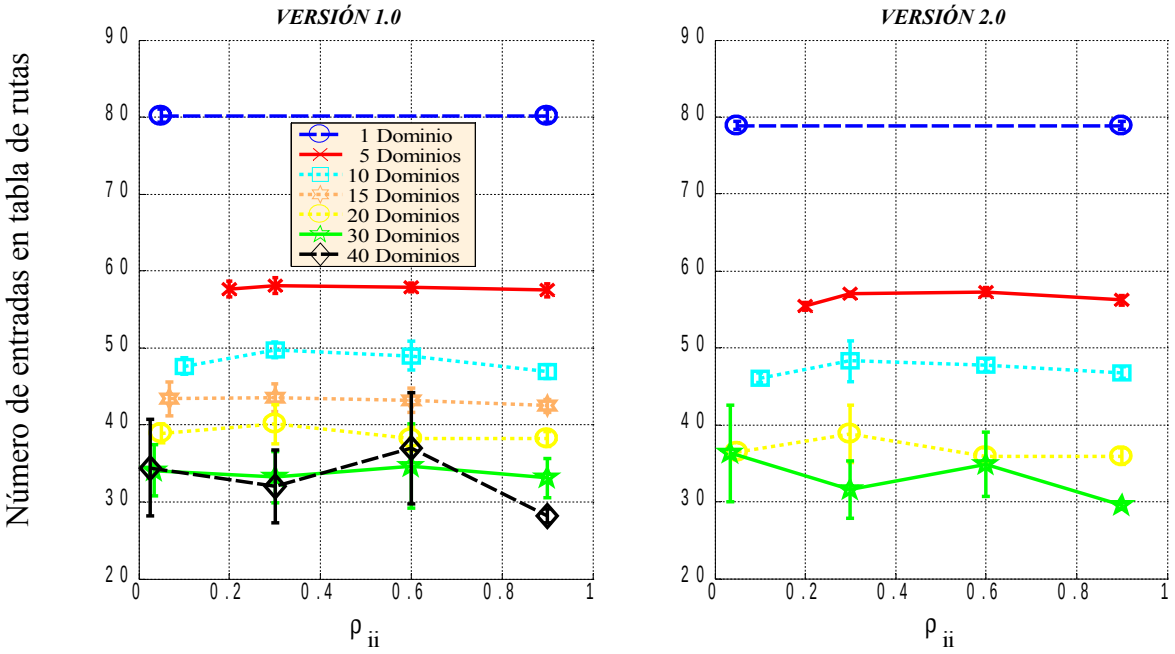


Figura 4.17: Número máximo de entradas en la tabla de rutas de los peers

Al igual que para el número medio de entradas, la tendencia de que teniendo un mayor número de dominios el número de entradas desciende se sigue cumpliendo para este caso, pero en sus valores máximos. También comentar que los valores entre ambas versiones son muy similares.

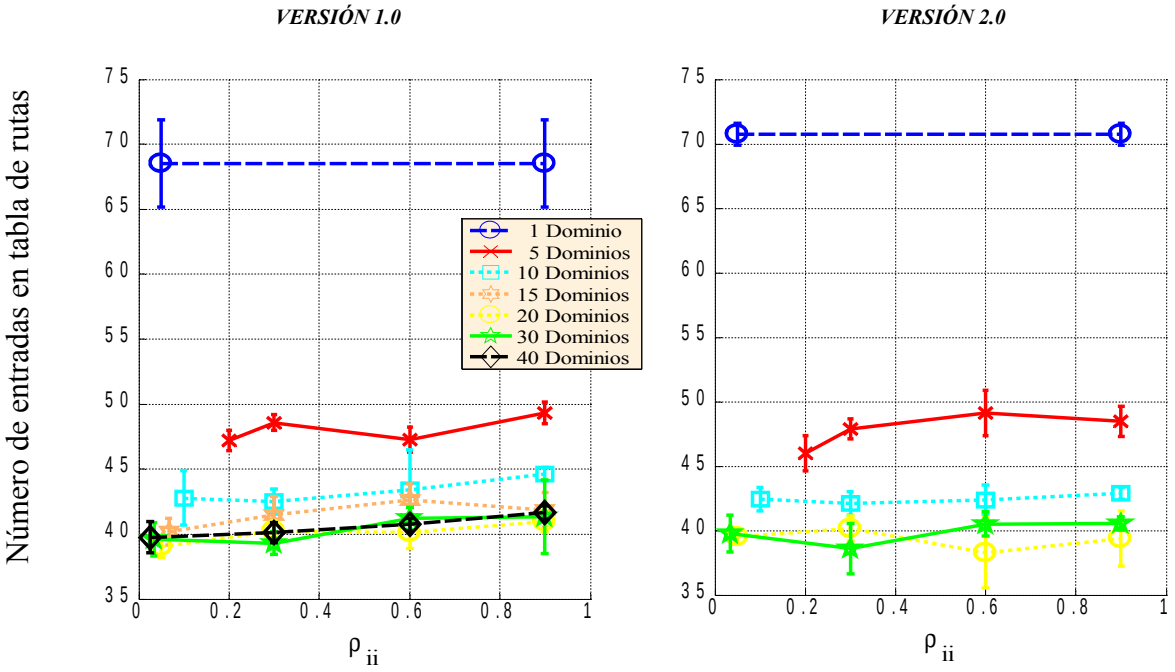


Figura 4.18: Número medio de entradas en tabla de rutas de superpeers

Para los superpeers sucede los mismo que para los peers y es que, a mayor número de dominios, el número de entradas en los superpeers disminuye porque al haber más dominios, hay mas superpeers y las búsquedas se dividen entre todos. Ambas gráficas muestran valores muy similares. Para valores altos de dominios los resultados no varían en exceso.

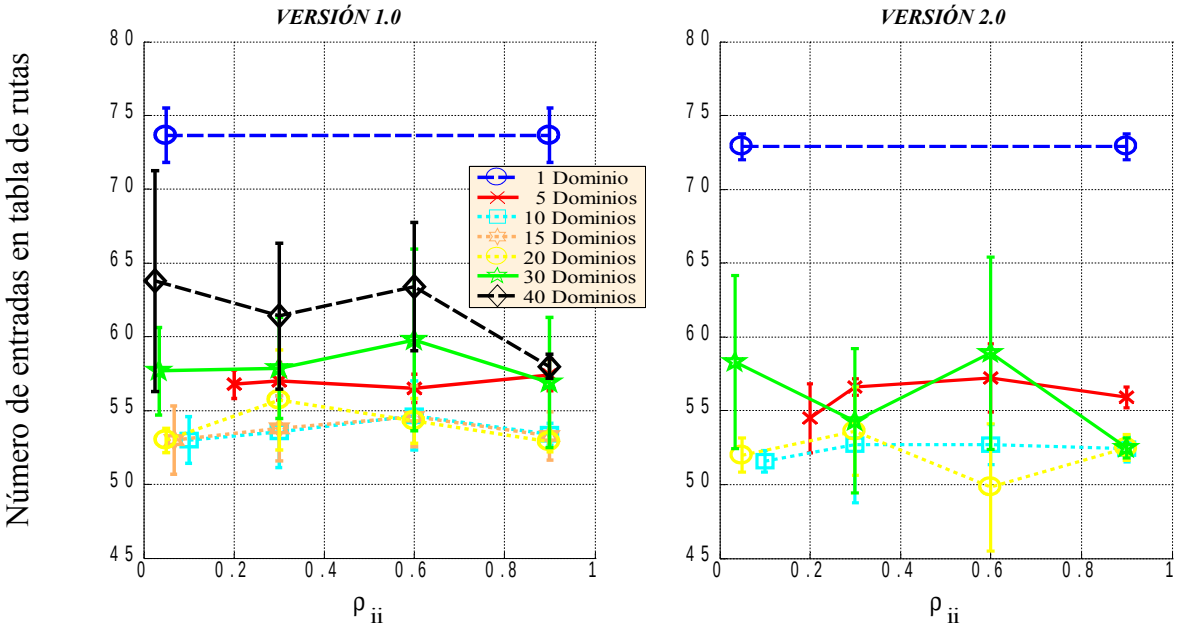


Figura 4.19: Número máximo de entradas en tabla de rutas de superpeers

Ambas gráficas muestran en sus valores máximos el mismo modelo y únicamente se puede destacar, que como es lógico, cuando sólo existe un dominio; es decir, tenemos un sólo superpeer, todas las búsquedas pasarán por él, aumentando su carga computacional.

3 Conclusión de la herramienta desarrollada

Para finalizar con este capítulo hay que añadir que, aunque parezca que este apartado pierde importancia frente al siguiente que son los resultados nuevos, no es así, debido a que si los resultados obtenidos en esta fase de validación no hubieran sido similares a los de la aplicación de partida, se podría decir que, la aplicación desarrollada para este proyecto no sería válida; en tanto en cuanto, los resultados del siguiente capítulo no serían significativos ni válidos como guía para futuros experimentos y modelos.

No obstante y a la vista de las gráficas expuestas en este capítulo, la aplicación desarrollada cumple perfectamente con los objetivos marcados al inicio, que son los ya mencionados, de por un lado, desarrollar una aplicación para poder simular redes reales que parte de otra muy compleja, pero todo ello reduciendo al mínimo la dificultad para configurarla y ejecutarla rápidamente, únicamente cambiando algunos parámetros de configuración en el XML del que se parte. Hay que decir que esto es la innovación más importante respecto a la aplicación de partida.

Capítulo V

GENERACIÓN DE NUEVOS RESULTADOS

1 Introducción

Este último capítulo, lo que va a mostrar son los resultados obtenidos al realizar nuevos experimentos con unos parámetros que con la aplicación de partida no se podían hacer, y que por tanto, añade al proyecto más valor aparte del ya mencionado; es decir, reducir la dificultad a la hora configurar la aplicación, pero utilizando toda la funcionalidad inicial. Por tanto, no existen de ellos ningún valor anterior, convirtiéndose así en una referencia para que programadores y desarrolladores puedan validar sus nuevos experimentos si deciden continuar por el camino que se ha iniciado en este proyecto.

En el capítulo anterior, en el que se validaba la aplicación diseñada se partía de que todos los dominios tenían el mismo número de peers y la colocación en dichos dominios podía ser determinista o aleatoria.

Pues bien, recordados los dos esquemas que se siguieron para validar la aplicación y con el fin de que se entiendan mejor los dos nuevos escenarios realizados, hay que decir en primer lugar que, el número de peers por dominio va a ser diferente, lo que añade mucho valor a este proyecto debido a que una de las limitaciones principales que tenía la aplicación de partida era precisamente que todos los dominios tenían el mismo número de peers.

Esta limitación se elimina gracias al nuevo diseño realizado y sobre todo a la introducción del archivo XML principal, lo que antes se podía ver como una limitación ahora es justamente lo contrario, convirtiéndose en uno de los puntos fuertes, porque en el archivo XML se puede especificar exactamente como va a ser cada dominio individualmente sin ningún tipo de restricción al respecto. Esta mejora no se ha realizado al azar, si no que se ha tomado como referencia un dato muy significativo y que todavía puede acercar más si cabe a un entorno real. Dicho dato es el número de usuarios de Internet en un determinado país.

La decisión de tomar este dato para configurar las nuevas pruebas viene dada por la temática de los experimentos y que mejor escenario posible que en el que los países con más usuarios de Internet puedan tener precisamente más nodos participantes en dichos experimentos, dotándolos así de mayor veracidad. Todos los datos usados en estos nuevos esquemas se han obtenido de la página web oficial de la CIA [CIA10a] y de otra página web que recoge las estadísticas de los usuarios de Internet por países [IWS10]. Por último, comentar que todos los valores que aparecen en el apéndice D se han redondeado para que siempre el numero de peers por experimento sea 1000.

Una vez comentado esto y a modo de conclusión, hay que señalar que las nuevas pruebas realizadas tendrán los nombres de distribución de forma determinista en entorno real donde se colocarán los peers de forma fija; y por otro lado, la distribución de forma aleatoria en entorno real donde podrá haber peers de un mismo dominio en distintos países.

2 **Resultados obtenidos**

Si bien, en el anterior capítulo se comentaba que los resultados mostrados en él eran muy importantes porque servían para validar la aplicación diseñada, en este, el objetivo es muy distinto y no es otro que demostrar que, variando los parámetros de configuración en el archivo XML sin utilizar mucho tiempo, se pueden obtener unos resultados totalmente nuevos y que serán una muy buena referencia para los desarrolladores que escojan simular sus escenarios con la nueva herramienta debido a que los resultados obtenidos se logran a partir de unos datos que representan el mundo real en cuanto a usuarios de Internet por país se refiere.

El orden de las gráficas que se mostrarán en este capítulo será el mismo que en el anterior para guardar una cierta lógica, este es: número medio y máximo de saltos según su probabilidad, número medio y máximo de saltos según el número de dominios, tiempo medio y máximo en realizar una búsqueda según la probabilidad, número medio y máximo de entradas en tabla de rutas de peers según la probabilidad y el número medio y máximo de entradas en tabla de rutas en superpeers según la probabilidad.

2.1 **Distribución de forma determinista en entorno real**

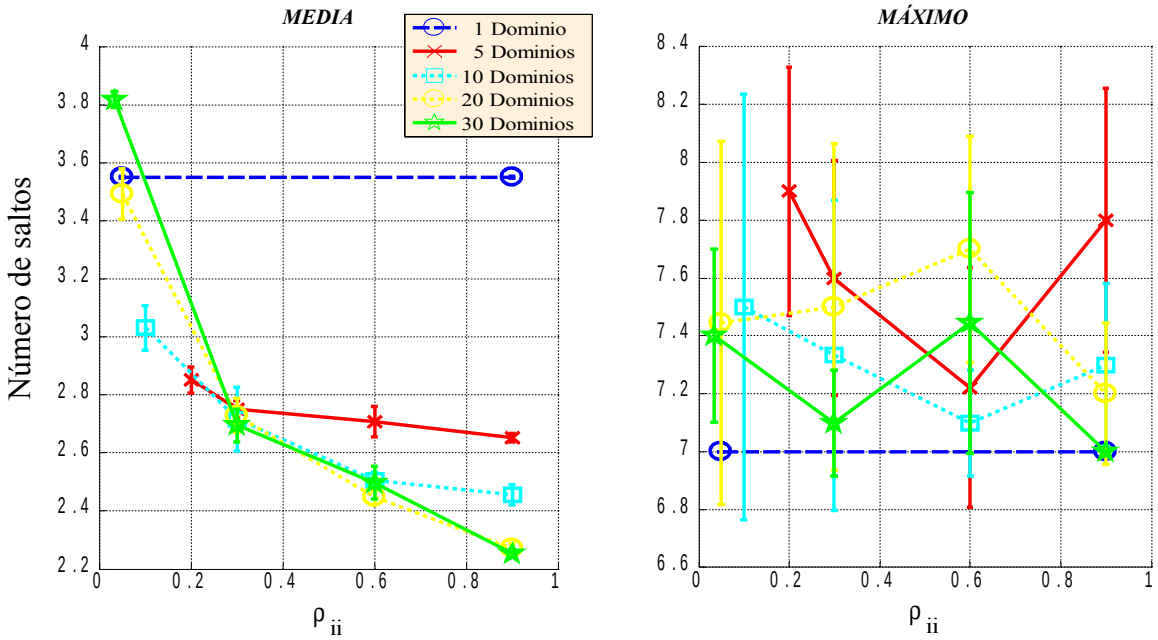


Figura 5.1: Número de saltos medios y máximos según la probabilidad

Cabía esperar estos resultados, ya que al igual que en el anterior caso donde todos los dominios tenían el mismo número de peers, a medida que aumenta el número de dominios y la probabilidad de buscar en el propio domino el número de saltos para realizar la búsqueda va disminuyendo independientemente del número de peers por dominio, salvo para los casos de 20 y 30 dominios donde se puede apreciar un cambio de modelo que se explicará en la siguiente gráfica.

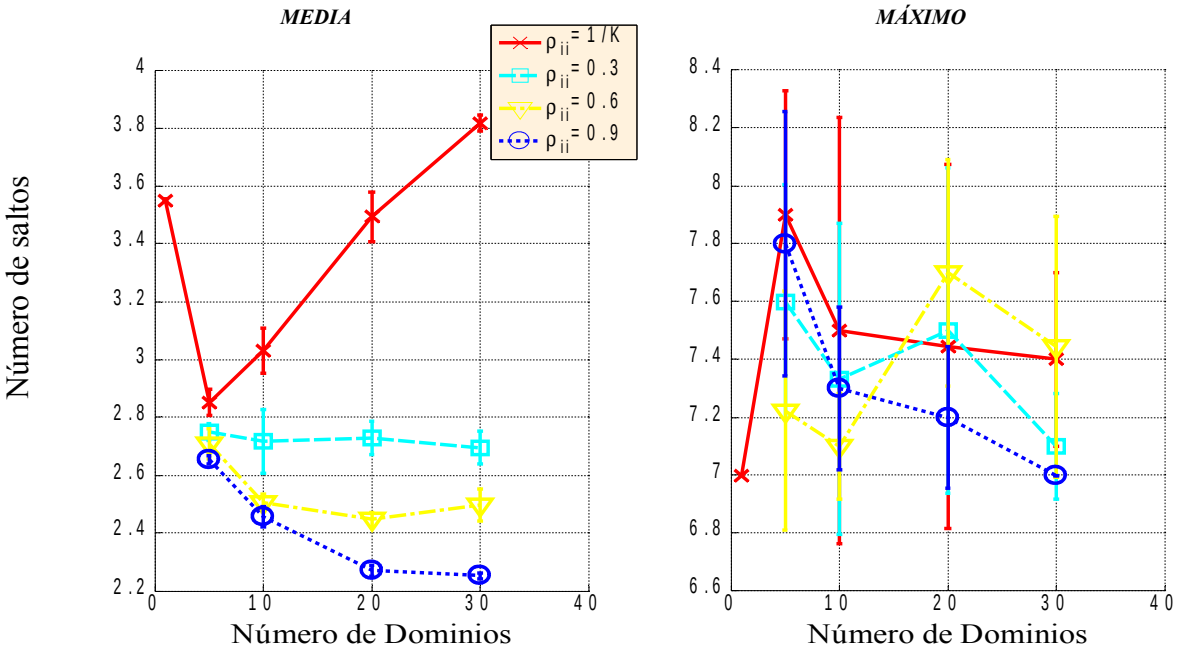


Figura 5.2: Número de saltos medios y máximos según el número de dominios

Cambiando el eje de abscisas de la probabilidad al número de dominios, en este caso de puede observar que el número de saltos va disminuyendo a medida que crece la probabilidad de buscar en nuestro dominio, pero en cambio, para las probabilidades 0.3, 0.6 y 0.9 de los dominios 20 y 30 son muy parecidas incluso aumentando en algún caso, lo que se puede dar por el hecho de que la introducción de 10 dominios nuevos dependiendo del número de usuarios de Internet no tenga el mismo peso que los 20 dominios iniciales, ya que el tráfico generado entre estos nuevos países no sea lo suficientemente significativo como para reducir el número de saltos en realizar una búsqueda. Para el caso donde la probabilidad es 1/K la tendencia sigue un razonamiento inverso al anterior, lo cual es lógico debido a que cuanto mayor es el número de dominios, menor será la probabilidad de buscar en el propio dominio lo que resultará en un mayor número de saltos para realizar una búsqueda satisfactoriamente.

De la gráfica con el número máximo de saltos hay que decir que no es especialmente significativa y de ella se puede extraer la conclusión de que el número máximo de saltos en este tipo de esquema es casi independiente al número dominios porque depende mucho de los países que participan en el experimento.

La gráfica 5.3 es una de las importantes y de la que se puede extraer mucha información. De ella se puede decir que el tiempo va disminuyendo a medida que la probabilidad de buscar en el propio dominio es mayor, porque la mayoría de las búsquedas se realizarán a peers del propio país y en este caso determinista, también del mismo dominio. En cambio, si se comparan estos resultados con los que se mostraban en el capítulo de validación, se pueden sacar varias conclusiones.

El tiempo va aumentando a medida que aumenta el número de dominios participantes en la prueba pero para los casos con 20 y 30 dominios se puede ver que los resultados son muy similares y esto puede ser debido a lo comentado en las anteriores gráficas; es decir, a que la introducción de 10 países nuevos para realizar los experimentos con un bajo número de usuarios de Internet y por tanto un bajo porcentaje de participación en los 1000 nodos, lleve a que los resultados con 20 y 30 dominios sean muy parecidos porque la probabilidad de que un peer de uno de los nuevos países participe en el experimento es baja debido a que de los 1000 nodos, los nuevos países tienen muy poco peso.

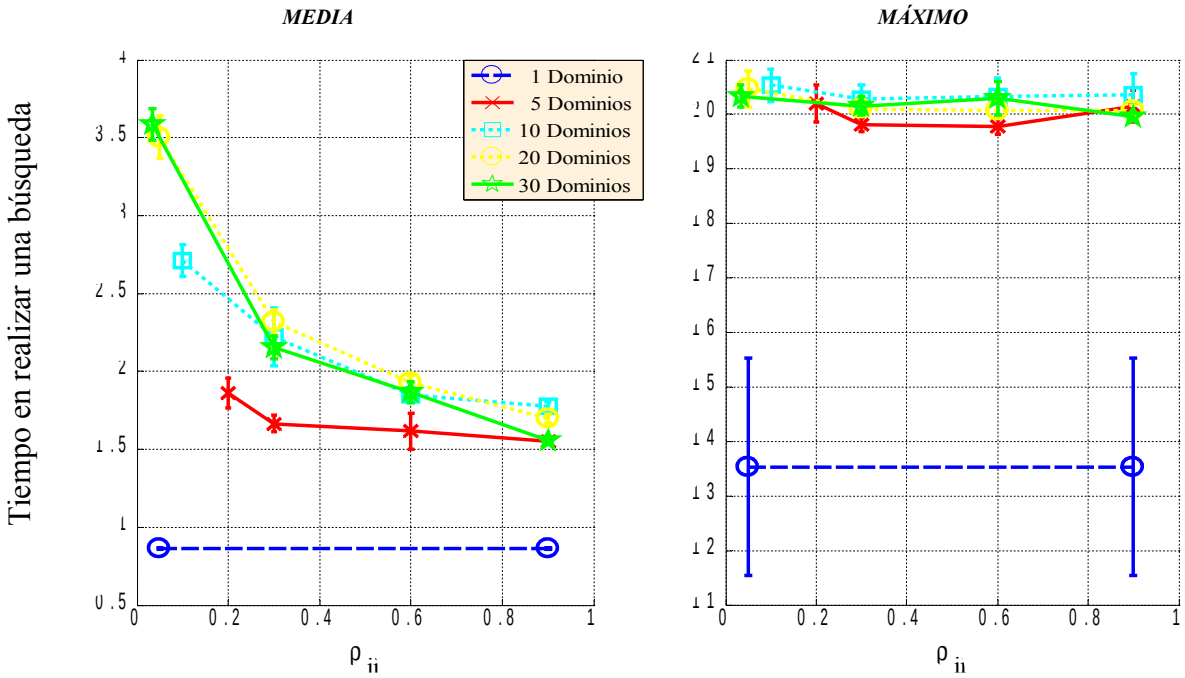


Figura 5.3: Tiempo medio y máximo en realizar búsqueda según la probabilidad

En cuanto a la gráfica del tiempo máximo en realizar una búsqueda hay que destacar que para números de dominios distintos a uno, el tiempo máximo empleado en realizar una búsqueda es similar, mientras que para un dominio evidentemente es mucho menor, lo cual también sobrecarga menos los equipos utilizados. Estos resultados se asemejan mucho al caso que se expuso en validación y es normal porque el nivel de sobrecarga de los equipos es similar en ambos escenarios.

Una vez vistos los valores de las querys para este escenario, se va a pasar a analizar las tablas de rutas de los nodos para este modelo determinista en entorno real.

Los valores mostrados en la figura 5.4 se asemejan mucho a los obtenidos para la validación, lo cual es lógico. El número de entradas en las tablas de los peers va disminuyendo a medida que aumenta el número de dominios porque existen más opciones de elegir un peer desde un país u otro, sobrecargándose estos menos. Los valores máximos como no podía ser de otra forma, sigue la misma tendencia que para los valores medios, a más dominios, menos entradas en cada tabla de ruta de peer.

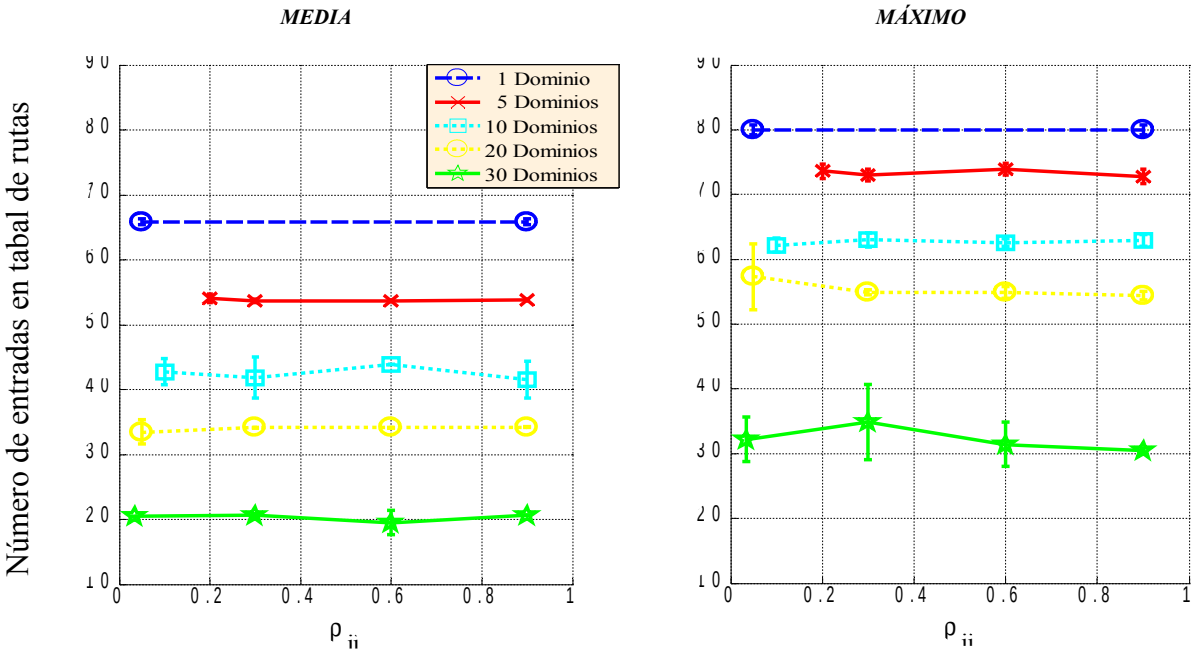


Figura 5.4: Número medio y máximo de entradas en la tabla de rutas de los peers

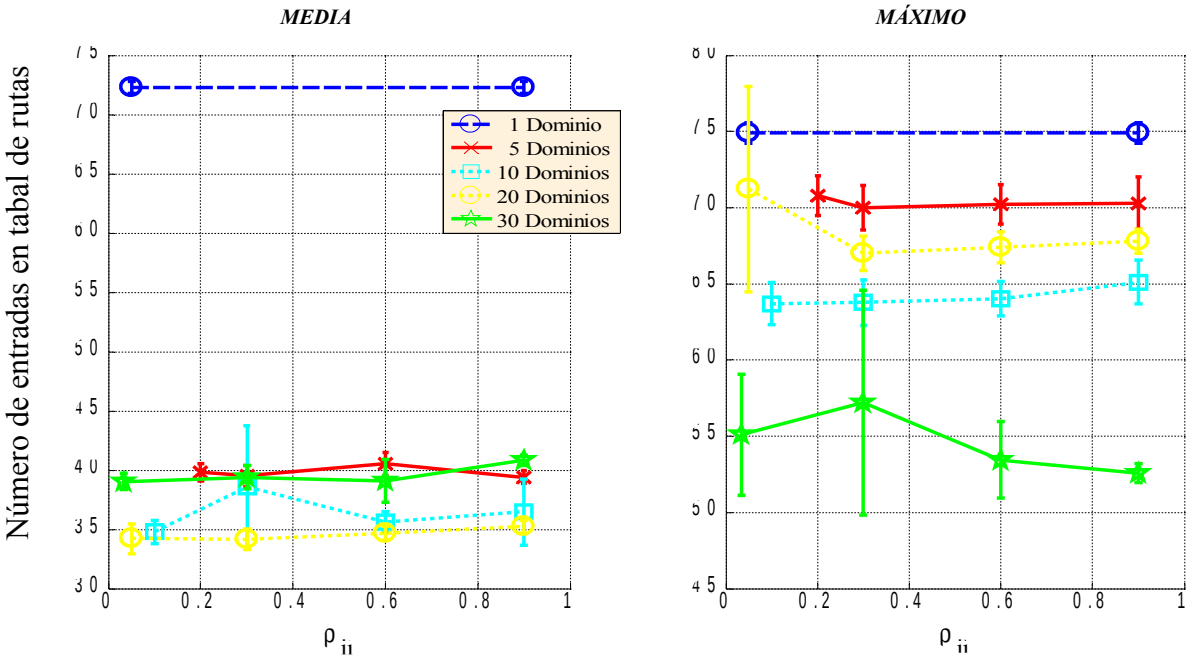


Figura 5.5: Número medio y máximo de entradas en la tabla de rutas de los superpeers

El caso de los superpeers es algo diferente al de los peers, ya que, para dominios diferentes a uno, la disminución de entradas no es tan pronunciada como para los peers, siendo valores casi idénticos debido a que aún existiendo más superpeers en el experimento, la red de interconexión estará igual de sobrecargada realizando las búsquedas. En comparación con el caso de validación hay que decir que los valores son muy similares, lo cual es lógico debido a que los escenarios son prácticamente iguales en cuanto a la red de interconexión se refiere.

En cuanto a los valores máximos hay que decir que para 20 dominios son mayores que para 10 dominios, lo que se puede deber a que se han manejado un número de clusters excesivos y que para 30 dominios no sucede debido a que la introducción de los 10 nuevos países resta sobrecarga a los superpeers participantes en la prueba.

Una vez mostrados y comentados los datos más relevantes en cuanto al número de saltos, tiempo en realizar búsquedas y entradas en las tablas de los nodos se refiere, a continuación se van a mostrar otro tipo de gráficas obtenidas durante los experimentos que mostrarán el tráfico que se ha generado en la red.

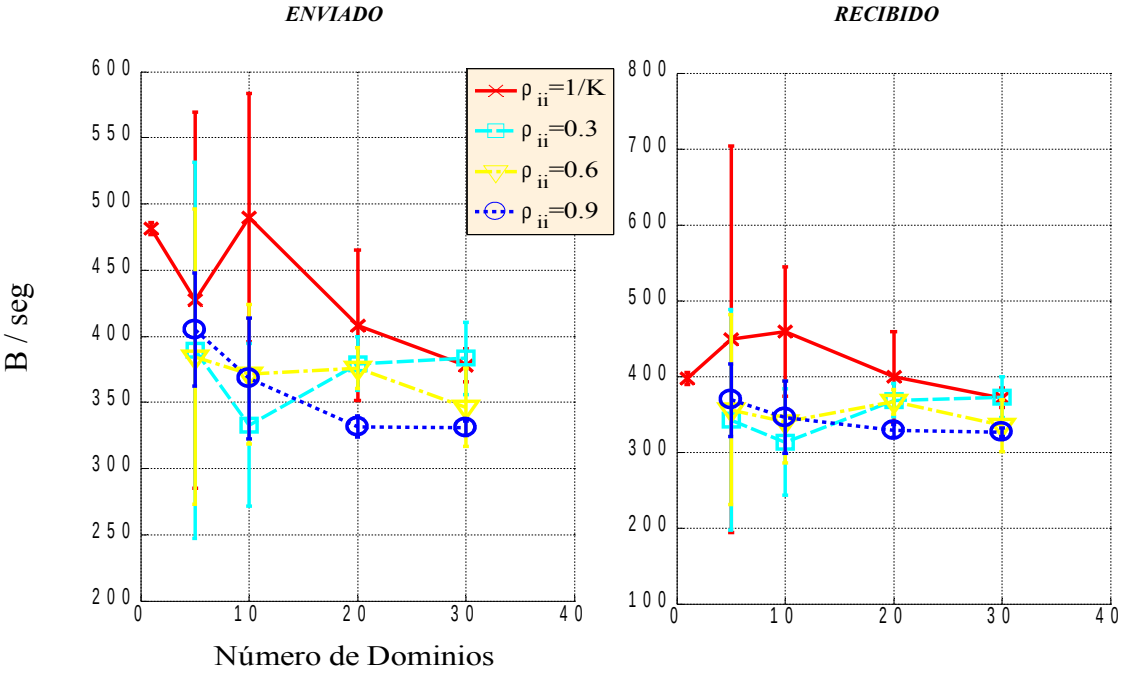


Figura 5.6: Tráfico medio enviado y recibido por los superpeers según el número de dominios

Viendo las gráficas hay que decir de ellas que para el tráfico enviado, no hay un modelo claro, pero si se puede ver que en la mayoría de los casos, a medida que aumenta el número de dominios, el tráfico que envía cada superpeer va disminuyendo. En cuanto al tráfico recibido por cada superpeer, el modelo aproximadamente es el mismo que para el tráfico enviado, lo cual es lógico debido a que para un mayor número de dominios, hay un mayor número de superpeers y se dividen entre todos el tráfico que se genera en la red de interconexión.

En las gráficas de la figura 5.7 se muestra el tráfico generado desde los superpeers a los peers de su dominio correspondiente, y lo que se puede ver en sus valores medios es que el tráfico va disminuyendo a medida que crece el número de dominios porque el número de superpeers va aumentando y cada uno se sobrecarga menos.

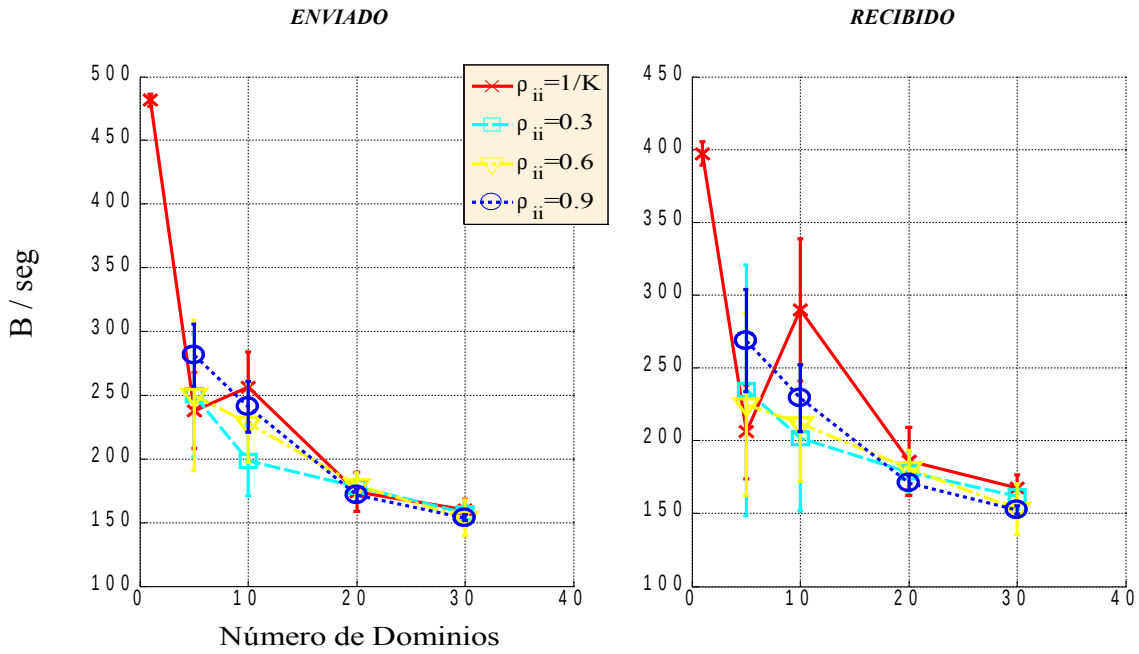


Figura 5.7: Tráfico medio enviado y recibido hacia abajo por los superpeers según el número de dominios

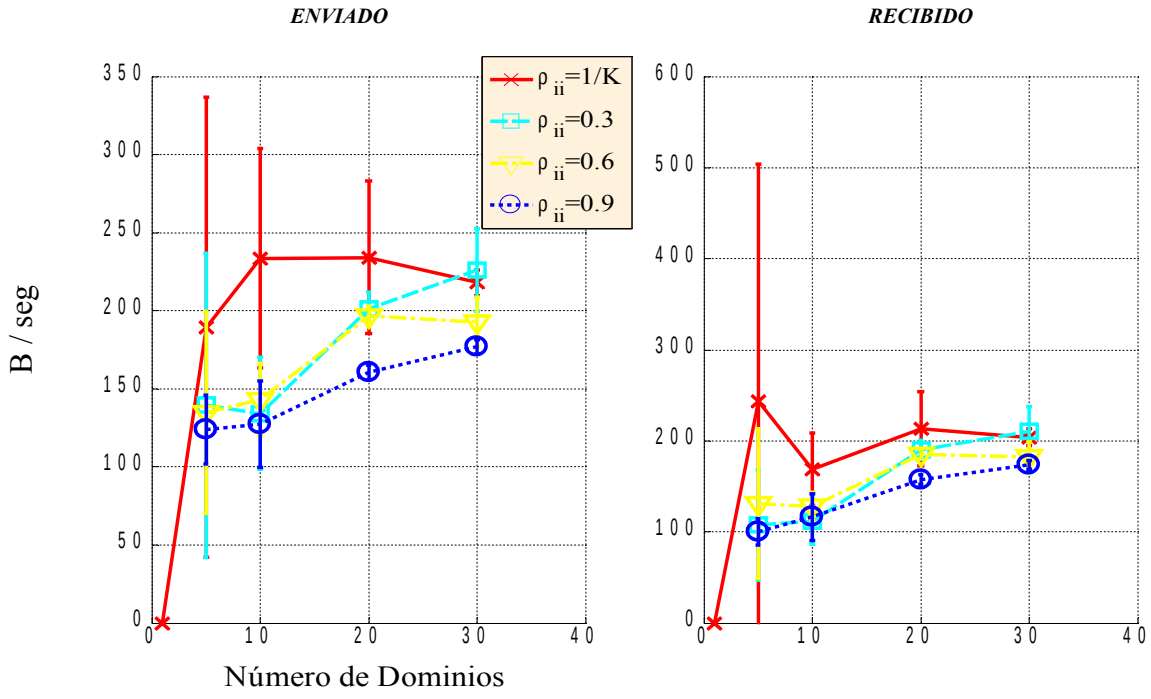


Figura 5.8: Tráfico medio enviado y recibido hacia arriba por los superpeers según el número de dominios

En este caso se está mostrando el tráfico que se genera en la red de interconexión entre los diferentes superpeers que participan en el experimento.

Se puede ver que para el caso de un dominio como es lógico no existe tráfico en la red de interconexión porque sólo hay un superpeer y todo el tráfico generado será entre ese superpeer y los peers de su dominio. En los valores medios de tráfico enviado se puede ver que va aumentando a medida que crece el número de dominios porque es más difícil encontrar un peer en un mayor número de dominios. Para los valores de tráfico recibido hay que destacar que el modelo seguido es el mismo pero el valor es menor porque cada superpeer cuando el número de dominios crezca, recibirá menos peticiones de búsqueda en su dominio.

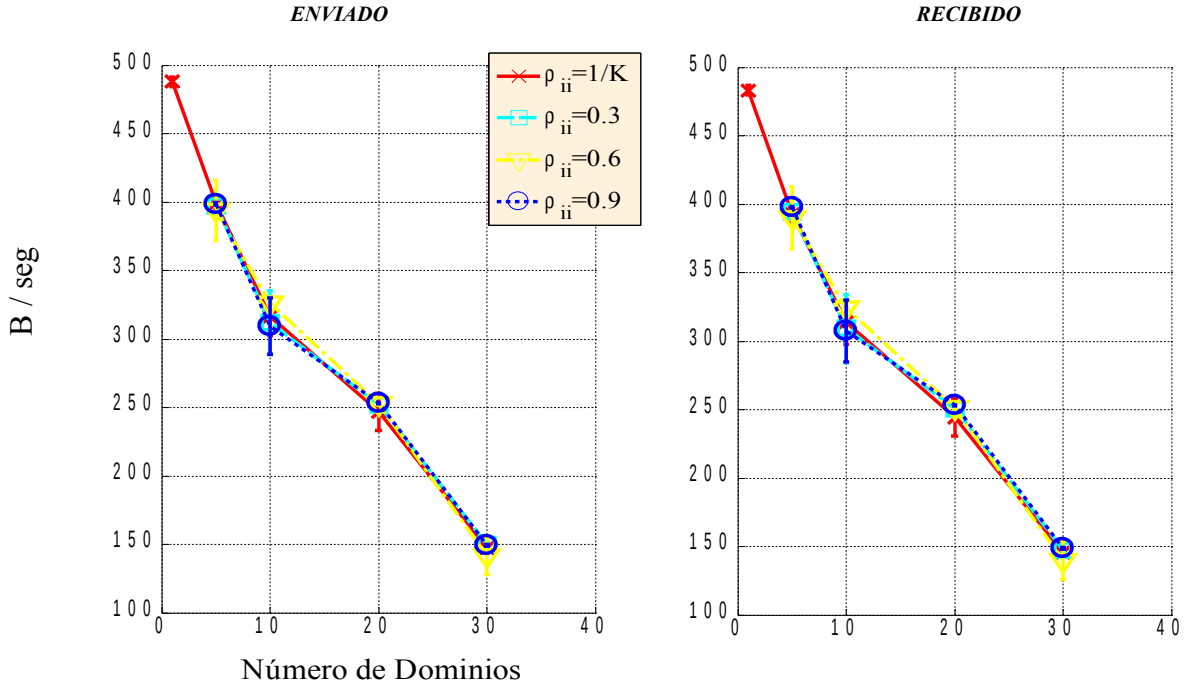


Figura 5.9: Tráfico medio enviado y recibido por los peers según el número de dominios

Por último y una vez visto el tráfico que se genera en la red de interconexión entre los diferentes superpeers, se va a mostrar el tráfico que se genera a nivel de cada dominio; es decir, el tráfico que envían y reciben los peers. Para ambos casos el modelo seguido es el mismo y los valores muy similares, a medida que crece el número de dominios, el tráfico enviado y recibido por cada peer que participa en el experimento va disminuyendo.

2.1 **Distribución de forma aleatoria en entorno real**

Para finalizar con este capítulo y con la explicación de los resultados obtenidos en esta fase del proyecto, se van a mostrar las mismas gráficas que para el caso determinista. También se comentarán dichas gráficas al igual que en el caso anterior siguiendo el mismo orden.

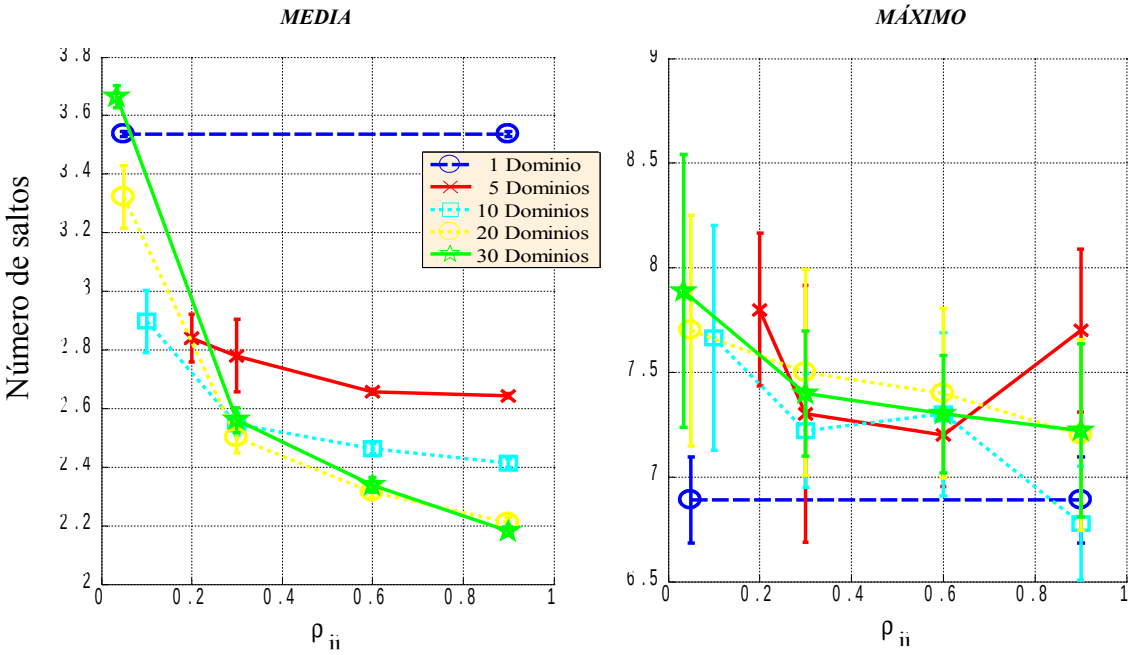


Figura 5.10: Número medio y máximo de saltos según la probabilidad

El modelo es similar al del caso determinista observando que para 20 y 30 dominios los valores obtenidos son muy parecidos debido a la introducción de países poco influyentes en los experimentos. Se podría pensar que los datos para este escenario tendrían que variar notablemente si se piensa que ahora, se tienen peers del mismo dominio en distintos países, pero precisamente este tipo de redes agrupa los peers por dominios independientemente de donde se encuentren y por tanto, el número de saltos realizados para hacer una búsqueda dentro de la red no se verá afectado.

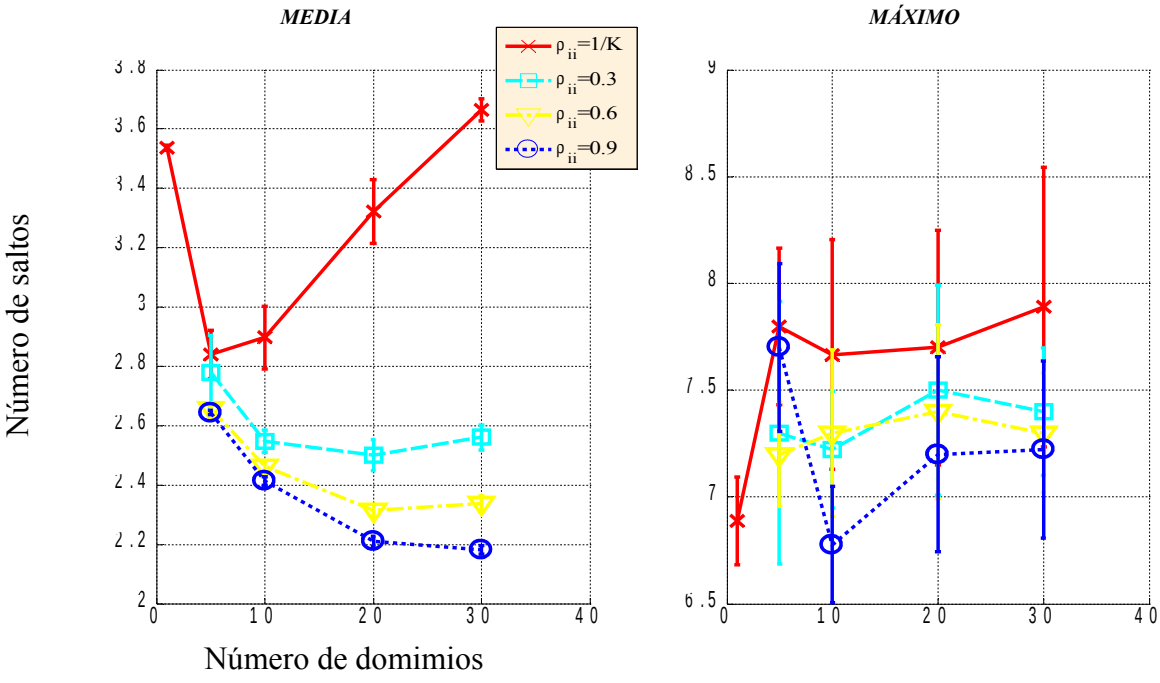


Figura 5.11: Número medio y máximo de saltos según el número de dominios

El razonamiento realizado en la figura 5.10 es el mismo que para la figura 5.11, únicamente varía el eje de abscisas.

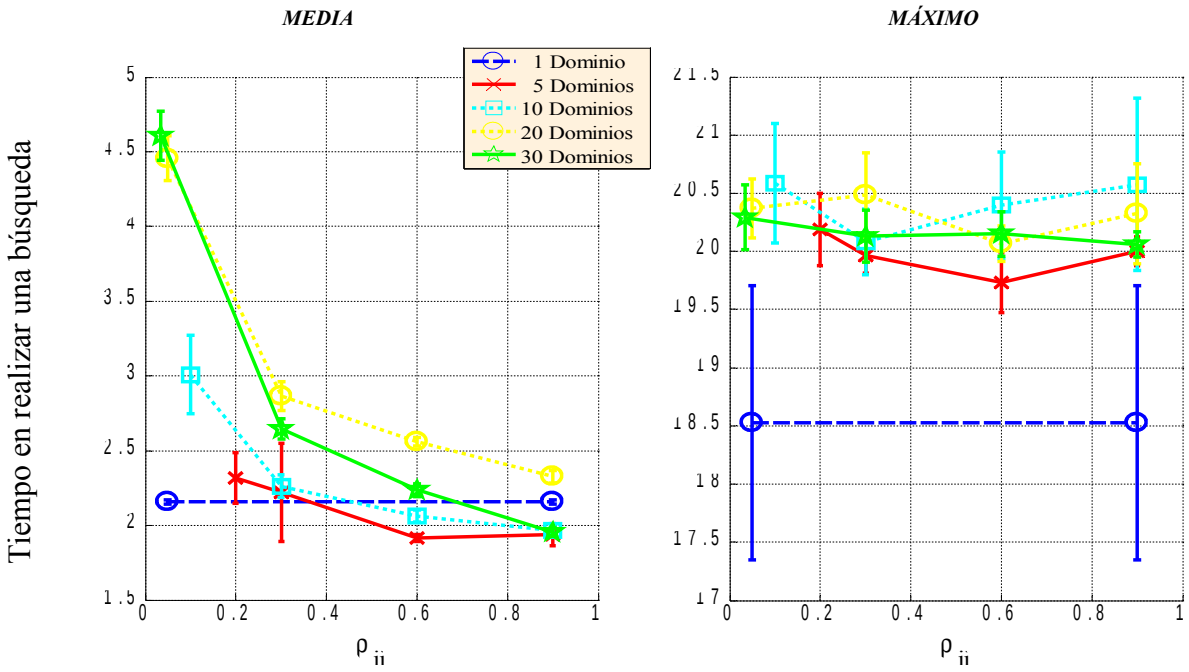


Figura 5.12: Tiempo medio y máximo según la probabilidad

De los valores que se muestran en la gráfica del tiempo hay que destacar que la tendencia lógicamente es la descender el tiempo en realizar una búsqueda a medida que la probabilidad de buscar en el propio dominio y el número de dominios aumenta. Pero en este caso, al contrario que en el número de saltos si es totalmente dependiente de los países que participen en la prueba debido a que al haber peers del mismo dominio en países diferentes, aún realizando una búsqueda dentro del propio dominio, el peer buscador y buscado pueden estar en países muy alejados, lo que sin duda influirá para que el tiempo en concluir dicha búsqueda sea mayor que para el caso determinista, dándose valores más llamativos.

En la figura 5.13 se puede ver que la variación de estos valores respecto al caso determinista tampoco es muy notable, lo cual es lógico debido a que el tráfico; y por tanto, el número de entradas en los peers que se producen es igual que para los escenarios anteriores.

Una vez más, en la figura 5.14 se puede ver que el modelo seguido es el de descender a medida que aumenta el número de dominios y esta vez también se puede ver como para 20 y sobre todo para 30 dominios los superpeers se sobrecargan teniendo más entradas que para el caso de 10 dominios.

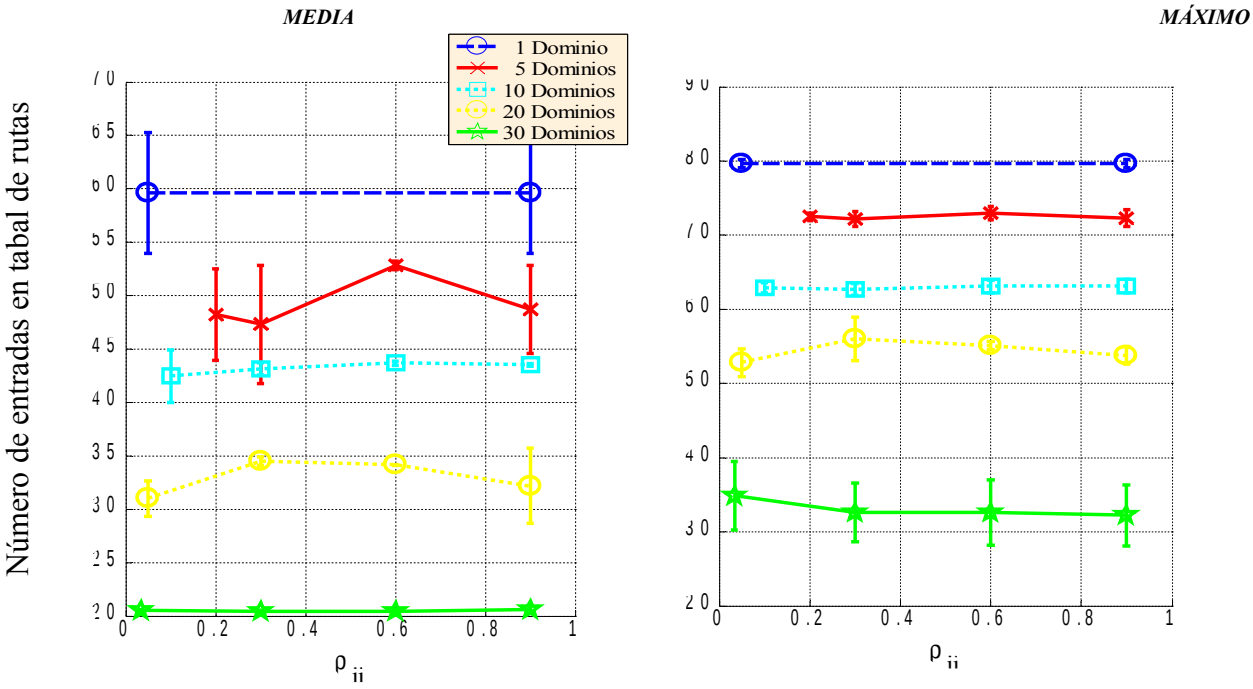


Figura 5.13: Número medio y máximo de entradas en la tabla de rutas de los peers

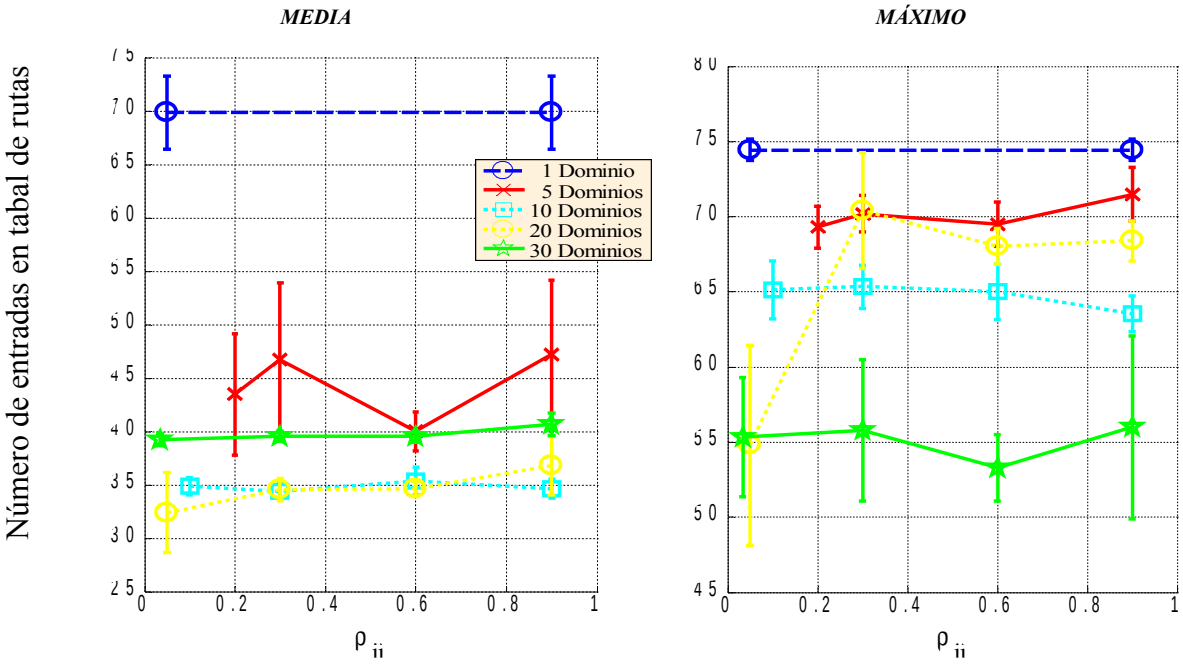


Figura 5.14: Número medio y máximo de entradas en la tabla de rutas de los superpeers

Una vez mostrados los datos más relevantes en cuanto al número de saltos, tiempo en realizar búsquedas y entradas en las tablas de los nodos, a continuación se van a mostrar las gráficas del tráfico generado durante los experimentos.

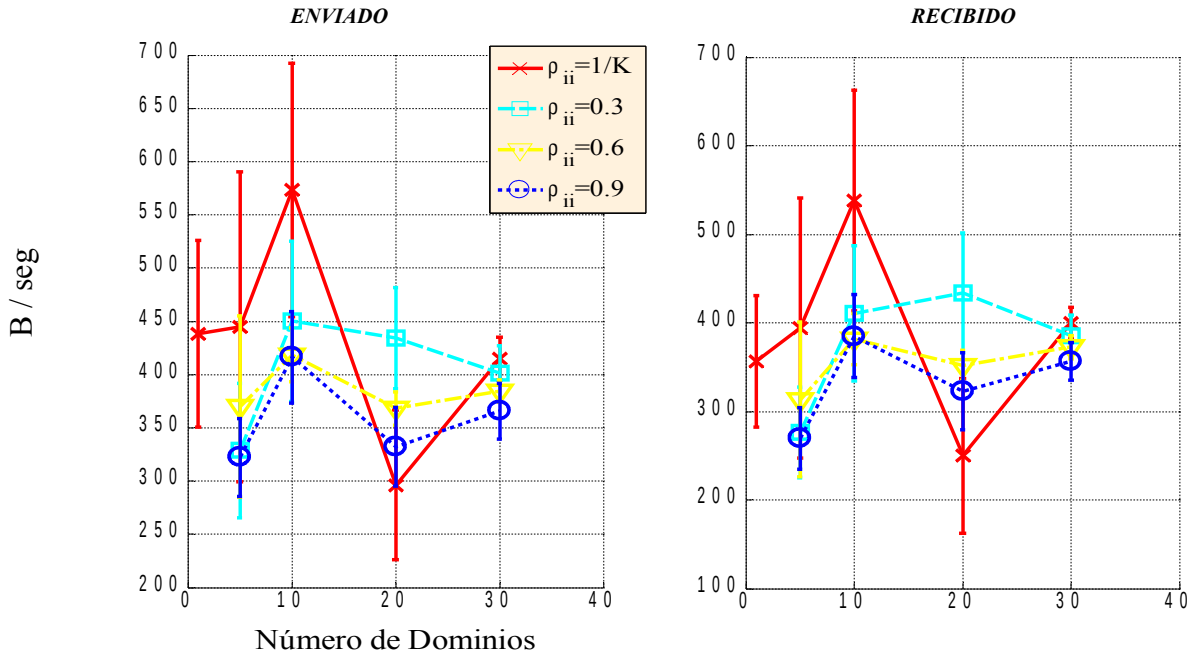


Figura 5.15: Tráfico medio enviado y recibido por los superpeers según el número de dominios

Como se puede ver en estas gráficas, en la red de interconexión, los superpeers no siguen ningún tipo de modelo que se pueda identificar, al igual que sucedió para el caso determinista.

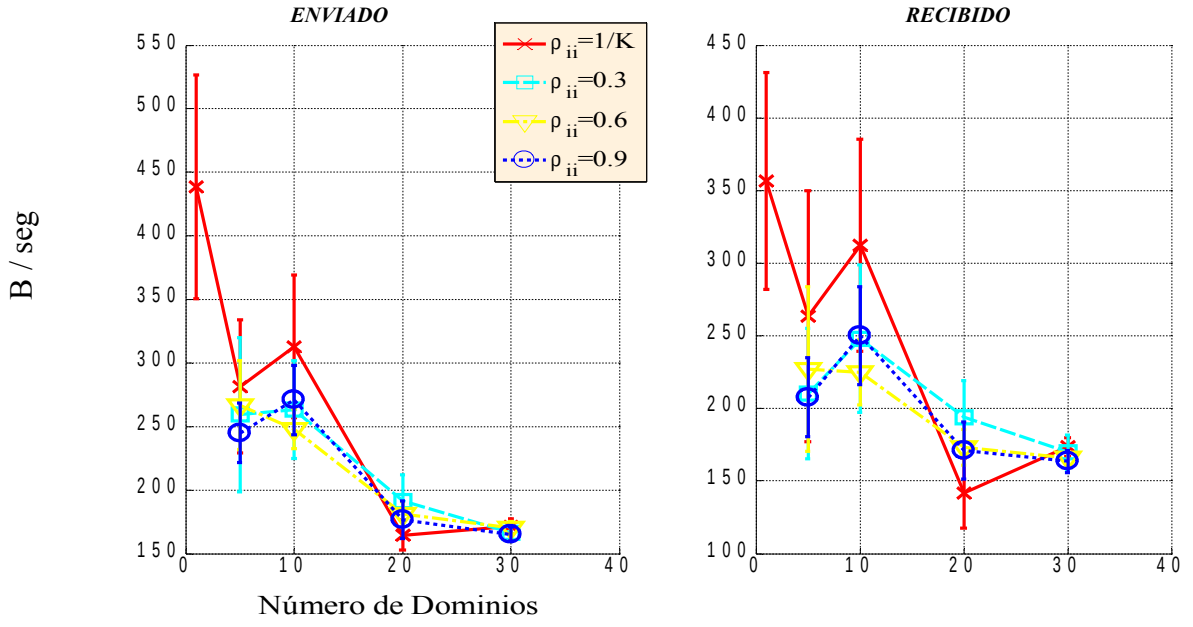


Figura 5.16: Tráfico medio enviado y recibido hacia abajo por los superpeers según el número de dominios

Al igual que para el caso determinista, las gráficas muestran la tendencia que cuando hay más dominios el tráfico intercambiado entre los superpeers y los peers de su dominio desciende porque existen más dominios participantes en las pruebas.

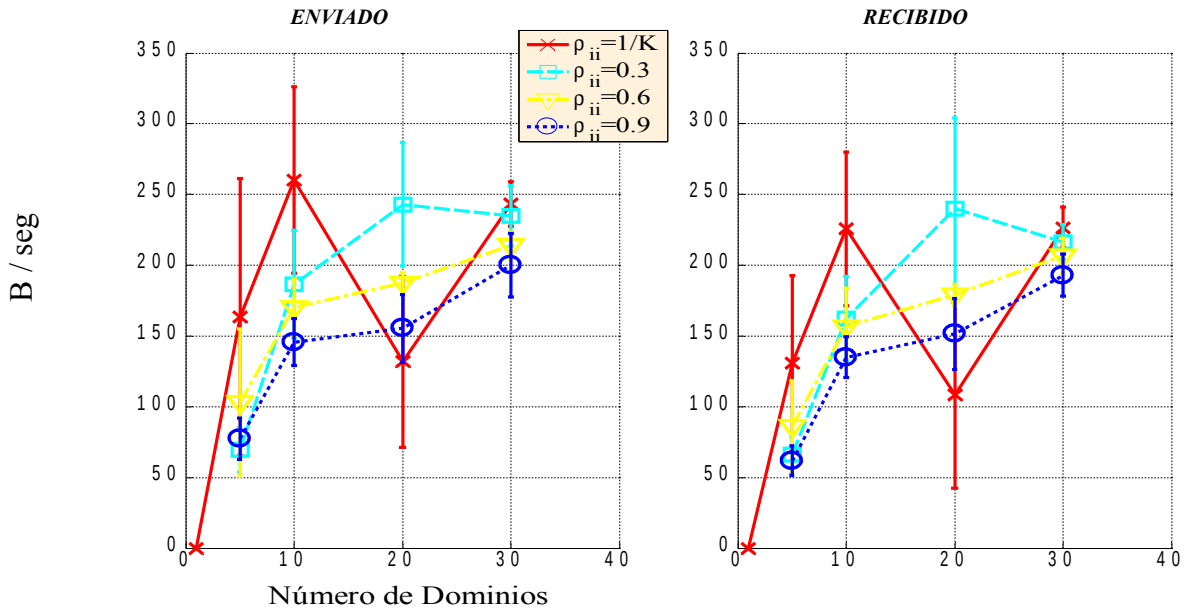


Figura 5.17: Tráfico medio enviado y recibido hacia arriba por los superpeers según el número de dominios

Al contrario que en las gráficas del tráfico entre superpeers y peers, ahora, el tráfico en la red de interconexión aumenta con el número de dominios debido a que al existir más dominios, la búsqueda es más compleja y los superpeers tienen que realizar más consultas para encontrar el superpeer al que pertenece el peer buscado.

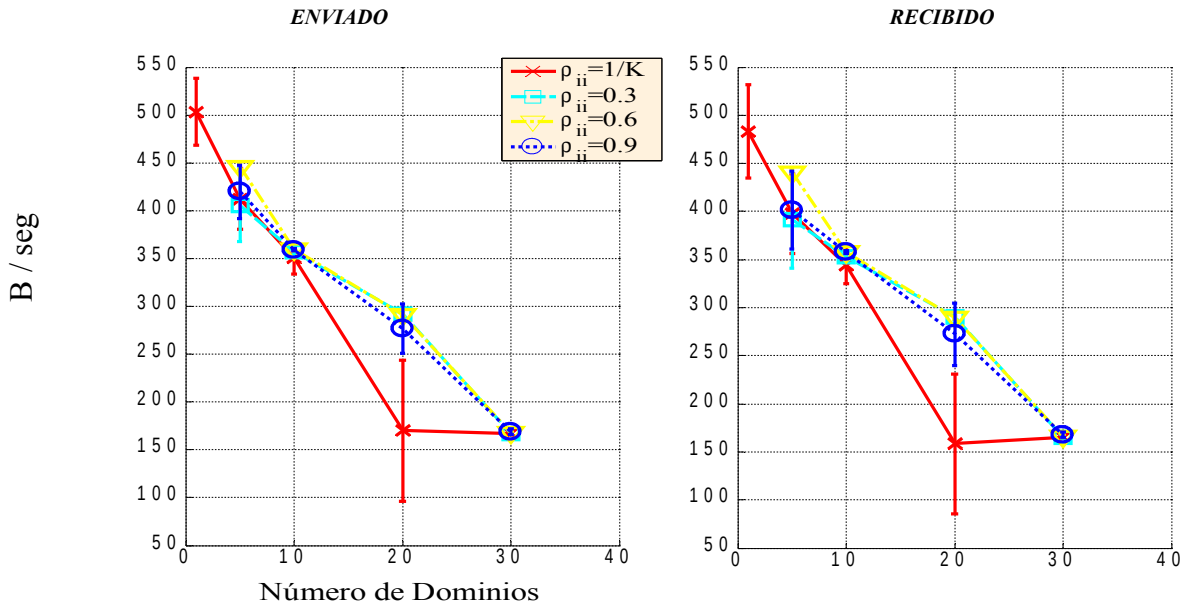


Figura 5.18: Tráfico medio enviado y recibido peers según el número de dominios

Esta última gráfica muestra que el tráfico generado por los peers va disminuyendo a medida que aumenta el número de dominios y esto es así porque como se ha visto en gráficas anteriores las búsquedas tardan más en realizarse y por tanto, el tráfico se concentra en la red de interconexión.

Para finalizar y una vez mostrados todos los resultados obtenidos, hay que decir que los mostrados en este capítulo siguen los modelos que se comentaron en el capítulo anterior y que valieron para validar el proyecto. Además se han añadido las gráficas más relevantes en cuanto al tráfico generado se refiere, lo cual añade más valor al proyecto, ya que anteriormente no existían resultados parecidos y que como se ha comentado en varias ocasiones, puede servir como punto de partida para los desarrolladores que elijan continuar con este trabajo y poder así validar sus aplicaciones con dichos resultados al igual que se ha hecho con la aplicación diseñada respecto a la de partida..

Capítulo VI

CONCLUSIONES

1 Conclusiones

Una vez finalizadas todas las pruebas y habiendo sido comentadas pertinentemente, a modo de conclusión hay que decir que la inclusión en el diseño del archivo XML ha sido una decisión acertada debido a que sin él, y como se ha visto en la aplicación de partida, se contaba con una herramienta muy potente para simular escenarios con redes P2P pero que a a la vez tenía una complejidad muy elevada a la hora de utilizarla; por tanto, perdía parte de ese potencial.

En cambio, al añadir el archivo XML para configurar las pruebas y todas las características de cada dominio individualmente se ha conseguido una herramienta con todo el potencial que tenía la inicial, pero reduciendo su complejidad de uso y configuración notablemente, lo que sin duda, dota a esta nueva herramienta de gran atractivo para los programadores y desarrolladores que sigan investigando en esta dirección.

Realmente se puede resumir el comentado potencial en algo tan sencillo como que configurando el archivo XML adecuadamente, se puede automáticamente desplegar una red en un entorno ModelNet e instalar en dicho entorno los programas necesarios en 1000 máquinas virtuales con tan sólo unos pocos pasos y poder así tener un escenario preparado para poner en funcionamiento distintos protocolos en redes overlay.

Una de las aplicaciones que se pueden usar son las de VoIP, que actualmente están en auge y que con esta herramienta pueden ser simuladas en el laboratorio con relativamente pocos equipos emulando entornos reales, tales como los que se usaron para las pruebas de la parte de generación de nuevos resultados del capítulo V.

En este apartado también hay que destacar que no sólo las conclusiones sobre la introducción del XML en la nueva herramienta y su uso es importante, si no que también, hay que destacar lo que se ha aprendido de todo el software utilizado en este proyecto.

En primer lugar hay que destacar el importante lugar que ha tenido el uso del emulador de red ModelNet, sin el cual no hubiera sido posible la realización de los escenarios utilizados para los experimentos. Tras multitud de pruebas ha que decir que se ha visto como una solución no muy escalable, lo cual es lógico, debido a que a más carga para el mismo número de routers da como resultado peor rendimiento. No obstante, encontrándose el compromiso entre un número de máquinas virtuales representativas y la sobrecarga del equipo, se ha visto que es un emulador muy fiable y estable.

Se ha comprobado también que gracias al software de virtualización Proxmox se pueden aprovechar de manera óptima los recursos hardware de los que se dispone en el laboratorio, pero sin olvidarse de los problemas asociados que también tiene esto, tales como el de compartir una misma tarjeta de red física para varios nodos virtuales.

Continuando con la parte técnica, hay que hacer mención especial a las tecnologías que han hecho posible la realización de esta herramienta, estas son Java y XML. De manera individual son muy potentes, pero para este proyecto ha sido de especial importancia las maneras que hay de utilizarlas conjuntamente, como por ejemplo JDOM, que ha sido la elegida.

Gracias a JDOM, se ha realizado la tarea de convertir, los datos contenidos en una etiqueta del archivo XML introducidos por el programador en, los parámetros que han usado las diferentes clases Java que componen las herramientas Despliegue de la red e Instalación, y que hacen posible la tarea de automatizar la instalación y puesta a punto de 1000 nodos virtuales, para ser usados en la ejecución de experimentos.

Por último, comentar que también se ha trabajado con scripts shell en diferentes sistemas operativos Linux, como son Ubuntu 8.04 (máquina virtual de control), FreeBSD 4.8 (máquina física y máquinas virtuales que funcionan como core, Biogridnet6, core1 y core3) y Debian Lenny (máquina física donde se ha desarrollado todo el software, Biogridnet2 y máquinas virtuales que funcionaban como edges nodes, edge2, edge3, etc). En todos ellos, y con la ayuda de ModelNet y Proxmox, se ha aprendido a instalar un cluster de máquinas virtuales, a gestionar un escenario con 9 máquinas virtuales y dentro de ellas, a emular una red con 1000 nodos virtuales, con los problemas que ello conlleva. En cuanto a la programación, las funciones de analizar, modificar y rediseñar un código creado por otra persona, adaptándolo a las necesidades del nuevo proyecto, también ha sido una función muy difícil y costosa personalmente, debido a que, la aplicación desde la que se partía para la realización de este proyecto, difería en muchos aspectos a los objetivos buscados en esta nueva versión.

2 Líneas futuras de trabajo

Debido a la naturaleza de este proyecto, durante el desarrollo del mismo y a la hora de ir realizándolo, han ido surgiendo otras opciones de diseño, que pueden verse como nuevas ideas y mejoras, transformándose en nuevos proyectos, continuando así con la línea trazada por este proyecto. A continuación se destacan algunos de ellos:

- Utilizar **otros emuladores de red** distintos a ModelNet para la realización de los escenarios de pruebas, tales como, Dummynet [DUM10], NISTNet [NIS10] , WANem [WAN10], todos ellos son software libre.

- Se puede extender la aplicación diseñada en este proyecto a un entorno real como **PlanetLAB** [PLA10], que es una red de investigación que soporta el desarrollo de nuevos servicios de red. En ella se pueden desarrollar nuevas tecnologías para almacenamiento distribuido, mapeo de red, sistemas peer-to-peer, tablas de hash distribuidas, y otras aplicaciones.

- Gestionar la **virtualización** de las máquinas con otro software, tales como el que proporciona Vmware [WMW10] y VirtualBox [VBO10], para poder administrar las máquinas virtuales donde se ejecutarán los experimentos.

- Realizar una **interfaz gráfica** que permita visualizar el escenario creado para cada experimento y poder administrar los diferentes parámetros simultáneamente a la ejecución de las pruebas.

APÉNDICES

A Tareas y memoria económica del proyecto

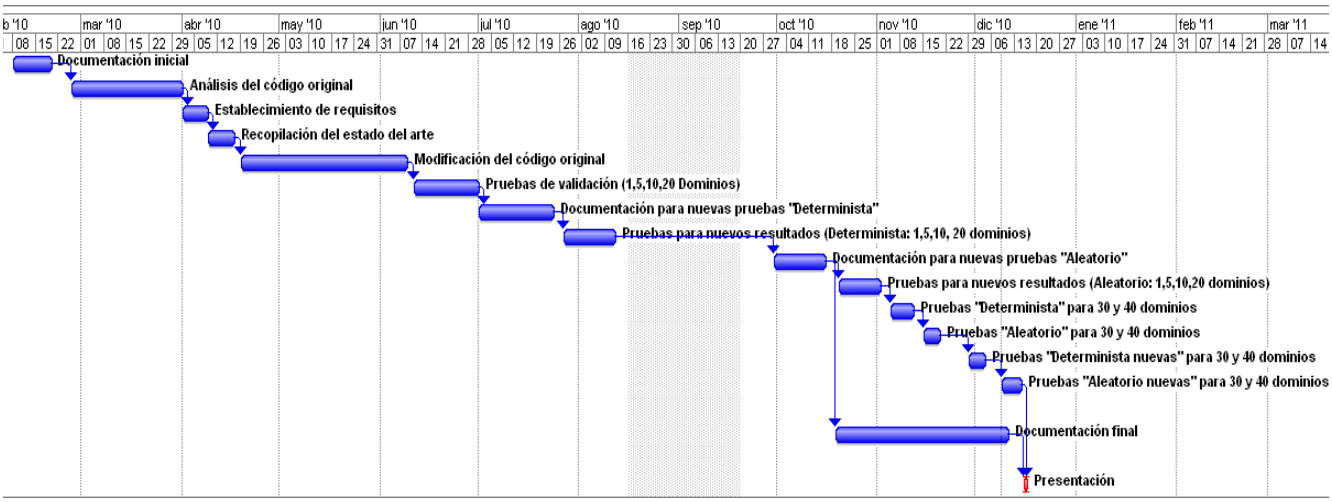
A.1 Tareas

Las tareas realizadas durante el proyecto son de diversa naturaleza y muchas de ellas se han ejecutado paralelamente a otras para que el tiempo total fuera el menor posible.

- Documentación inicial (2 semanas).
- Análisis del código original (1 mes).
- Establecimiento de requisitos (1 semana).
- Recopilación del estado del arte (1 semana).
- Modificación del código original (6 semanas).
- Realización de las pruebas de validación (1 mes).
- Documentación para realización de nuevas pruebas (1 mes).
- Realización de las pruebas para nuevos resultados (6 semanas).

Tareas paralelas:

- Documentación final (1 mes).



A.2 **Memoria económica**

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Félix Gómez Fernández
2.- Departamento:
Ingeniería Telemática
3.- Descripción del Proyecto:
- Titulo: **Automatización de instalación y configuración de aplicaciones peer-to-peer en un entorno virtualizado basado en ModelNet y Proxmos**
- Duración (meses): **10**

4.- Presupuesto total del Proyecto (valores en Euros):
20.543,33 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F	Categoría	Dedicación (meses)	Coste (mes)	Coste (Euros)
Gómez Fernández Félix	50210341-F	Ingeniero	10	1.800,00	18.000,00
				Total	18.000,00

EQUIPOS

Descripción	Coste (Euros)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Host A (4x2000)	8.000,00	100	14	60	1.866,67
Host B (2x1000)	2.000,00	100	14	60	466,67
Pantalla LCD (3x200)	600,00	100	14	60	140,00
Switch (1x200)	200,00	100	14	60	46,67
KVM (1x100)	100,00	100	14	60	23,33
					0,00
				Total	2.543,33

[d] Fórmula de cálculo de la amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	18.000
Amortización	2.543
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	0
Total	20.543

El presupuesto total de este proyecto asciende a la cantidad de 20.543 Euros.

Leganés a 16 de Diciembre de 2010 Fdo. Félix Gómez Fernández

B - **ModelNet**

Este apéndice ha sido extraído del aparecido en la versión de partida [SENB07a], y que por motivos de comodidad para el lector se ha introducido también en esta versión.

Introducción

ModelNet [MOD10] es un emulador de red de gran escala que nos permite evaluar sistemas distribuidos en un entorno realista similar a Internet. ModelNet permite probar prototipos sin necesidad de modificarlos sobre sistemas operativos normales en diferentes escenarios de red. Se puede decir, que combina la repetitividad de la simulación con el realismo de un desarrollo real. ModelNet ha sido desarrollado para ayudar en el diseño y prueba de las nuevas redes de distribución de contenido, sistemas P2P, protocolos de la capa de transporte, procesadores de flujos distribuidos, sistemas de archivos distribuidos y herramientas de medición de red.

Una vez desplegado ModelNet, cada instancia de la aplicación corre en un nodo virtual; ModelNet multiplexa los nodos virtuales entre el conjunto de las máquinas físicas disponibles que se llamarán "edge nodes". El sistema configura los edge nodes para enrutar sus paquetes a través del core de ModelNet (compuesto por uno o más máquinas físicas). Este core adapta cada paquete a su ancho de banda, tiempo de transito, perdidas… en la topología elegida. ModelNet soporta la emulación hop-by-hop, capturando los efectos del trafico cruzado y la congestión en la red.

Usar ModelNet es relativamente fácil y requiere los siguientes pasos:

- Especificación de la topología. Esta topología puede ser cualquiera, desde una red full-mesh hasta una topología real completa.

- Vinculación. ModelNet mapea los nodos virtuales en los edge nodes, dándole a cada uno una IP en el rango 10.0.X.X. En esta fase también se construye la "tabla de rutas" que contiene el conjunto de túneles a atravesar para cada par origen-destino.

- Despliegue. Por último se deben desplegar los archivos generados en los pasos anteriores a todas las máquinas del cluster. ModelNet necesita un mínimo de 2 máquinas para funcionar, pero lo puede hacer con más de 100.

Instalación

La instalación de ModelNet se basa en dos pasos claramente diferenciados: la instalación de los core nodes y la instalación de los edge nodes, completamente diferentes entre sí.

Instalación de los edge nodes

Para la instalación de los edge node en el sistema se necesita tener instalada previamente una distribución de Linux. Para este caso se ha elegido una Debian Lenny. Una vez instalado el sistema operativo solo hay que ejecutar el siguiente script:

```
#!/bin/bash

apt-get update

#apt-get install linux-headers-2.6.18-6-686

apt-get install openssh-server

apt-get install libxml-simple-perl

apt-get install libboost-graph-dev

apt-get install sudo

apt-get install libxerces27-dev

apt-get install make

apt-get install g++

apt-get install iperf

apt-get install rsync

#wget http://bittella.googlecode.com/files/modelnet-linux-0.99.tar.gz

wget http://bittella.googlecode.com/files/modelnet-0.99.tar.gz

wget http://bittella.googlecode.com/files/Graph-0.20105.tar.gz

wget http://bittella.googlecode.com/files/Heap-0.80.tar.gz

#wget http://bittella.googlecode.com/files/sstv_7_08_linux.patch

tar -xvzf Heap-0.80.tar.gz

cd Heap-0.80

perl Makefile.PL

make

make install

cd ..

rm -rf Heap-0.80

tar -xvzf Graph-0.20105.tar.gz
```

```
cd Graph-0.20105

perl Makefile.PL

make

make install

cd ..

rm -rf Graph-0.20105

tar -xvzf modelnet-0.99.tar.gz

cd modelnet-0.99

mkdir linux

cd linux

../configure

make

make install

cd ../../
```

Para ello hay que seguir los siguientes pasos:

1- Crear máquina en el vbmr1.
2- Cambiar /etc/dhc3p/dhclient.conf para añadir 'send host-name "hostname".
3- Ejecutar "ifconfig eth0 up".
4- ejecutar dhclient.
5- wget enjambre.it.uc3m.es/~rsanchez/ModelNet/configura.sh.
6- chmod 777 configura.sh.
7- ./configura.sh.
Dar a todo "Y".
8- Cambiar máquina a vbmr0 y reiniciar.

Instalación de los core node

La instalación de los core nodes es un poco más complicada que la de los edge nodes ya que en un primer paso hay que recompilar el kernel de un sistema operativo FreeBSD. Para este caso se ha elegido un FreeBSD 4.8.

Para facilitar estos pasos se han usado los siguientes scripts.

Install_frebsd.sh

```
#!/bin/csh

setenv PACKAGESITE ftp://ftp-archive.freebsd.org/pub/FreeBSD-Archive/ports/i386/

packages-4.8-release/Latest/

pkg_add -r wget gmake bash

pkg_add -r perl sudo

pkg_add -r p5-XML-Simple linuxthreads

pkg_add -r libgnugetopt

pkg_add -r boost

pkg_add -r xerces-c2
```

Install_frebsd2.sh

```
#!/bin/csh

wget http://bittella.googlecode.com/files/Graph-0.20105.tar.gz

wget http://bittella.googlecode.com/files/Heap-0.80.tar.gz

wget http://bittella.googlecode.com/files/modelnet-0.99.tar.gz

tar -xvzf Heap-0.80.tar.gz

cd Heap-0.80

perl Makefile.PL

gmake

gmake install

cd ..

rm -rf Heap-0.80

tar -xvzf Graph-0.20105.tar.gz

cd Graph-0.20105

perl Makefile.PL

gmake

gmake install

cd ..

rm -rf Graph-0.20105
```

```
tar -xvzf modelnet-0.99.tar.gz

cd modelnet-0.99

mkdir freebsd

cd freebsd

../configure --with-fbsdsys=/sys/

gmake

gmake install
```

Para ello hay que seguir los siguientes pasos:

1- Iniciar KVM(Qemu) con el Cd del FreeBSD 4.8 metido en vbmr1
2- Instalar para kern-development
3- Que sea Gateway y no compatible con los binarios de Linux.
4- Terminada la instalación reiniciar
5- recompilar kernel:

cd /sys/i386/conf/

ee GENERIC (añadir línea options HZ=10000)

config GENERIC

cd ../../compile/GENERIC

make (tarda un poco)

cp kernel /

6- reiniciar.
7- copiar archivos de instalación install e install2
8- chmod 777 instal*
9- ./install (2 veces por si acaso)
10- ./install2
11- ee /etc/rc.conf (cambiar nombre de dominio)
12- ee /etc/dhclient.conf (añadir send host-name "nombre";)
13- apagar y cambiar el vmbr1 por el vmbr0
14- encender
15- visudo

quitar el # de delante de root y wheel.

salir (Q salen los dos puntos y pones wq!)

16- ee /etc/inetd.conf (Quitar el comentario de login, finger y exec)
17- ee /etc/ssh/sshd_config

PermitRootLogin yes
RSAAuthentication yes

PubkeyAuthentication yes

AuthorizedKeysFile /root/.ssh/authorized_keys

18- reiniciar y debería estar

Construcción de una topología

Para ejecutar una red ModelNet hay que crear varios archivos XML:

• Gráfico (.graph) - las listas de los nodos y enlaces de la red virtual.

• Rutas (.route) - contiene la información para enrutar los paquetes a través de la red virtual.

• Máquinas (.hosts) - lista de las máquinas que actuarán como edge nodes y como core nodes.

• Modelo (.model) - lista de los nodos que se encuentran en cada máquina y de los enlaces entre ellos.

Para actuar ModelNet requiere el archivo de rutas y el archivo del modelo. El archivo con el gráfico y el que nos indica las máquinas disponibles solo se usa en la generación de los dos anteriores. ModelNet incluye herramientas en perl que pueden ser usadas para generar los archivos requeridos a partir del .graph y el .hosts.

Gráfico de la topología

El archivo .graph que hay que generar debe tener un formato similar al siguiente ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<topology>
<vertices>
<vertex int_idx="0" role="gateway" />
<vertex int_idx="1" role="gateway" />
<vertex int_idx="2" role="virtnode" int_vn="0" />
<vertex int_idx="3" role="virtnode" int_vn="1" />
<vertex int_idx="4" role="virtnode" int_vn="2" />
</vertices>
<edges>
<edge int_dst="1" int_src="2" int_idx="0" specs="client-stub" int_delayms="1" />
<edge int_dst="2" int_src="1" int_idx="1" specs="client-stub" dbl_kbps="768" />
<edge int_dst="1" int_src="3" int_idx="2" specs="client-stub" />
<edge int_dst="3" int_src="1" int_idx="3" specs="client-stub" />
<edge int_dst="0" int_src="4" int_idx="4" specs="client-stub" />
<edge int_dst="4" int_src="0" int_idx="5" specs="client-stub" />
```

```
<edge int_dst="1" dbl_len="1" int_src="0" int_idx="0" specs="stub-stub" />
<edge int_dst="0" dbl_len="1" int_src="1" int_idx="1" specs="stub-stub" />
</edges>
<specs >
<client-stub dbl_plr="0" dbl_kbps="64" int_delayms="100" int_qlen="10" />
<stub-stub dbl_plr="0" dbl_kbps="1000" int_delayms="20" int_qlen="10" />
</specs>
</topology>
```

En él se distinguen 3 partes claramente diferenciadas:

- **Vertices.** En esta sección hay que añadir todos los nodos virtuales que va a tener la topología, tanto los nodos que formarán el núcleo de la red como los que actuarán de nodos virtuales.

- **Edges**. En esta sección se definen los enlaces de la topología.

- **Specs**. Por último, se definen las especificaciones para cada tipo de enlace. Si un enlace específico tiene información sobre su ancho de banda, retardo o pérdidas, esta información sobrescribirá a la que se encuentre en su tipo.

Para la generación de este archivo se puede usar herramientas como inet2xml o generarlo directamente a mano.

Archivo con las máquinas disponibles

El archivo .hosts que hay que generar tiene un formato similar al siguiente ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<hardware>
<emul hostname="larry"/>
<host hostname="curly"/>
<host hostname="moe"/>
</hardware>
```

Como se ve en este sencillo archivo hay que listar el nombre de las máquinas que se van a usar. Las que se definen como “emul” serán usadas como core nodes y las que se definen como “host” serán las que hagan la labor de edge node.

Generación del archivo de rutas y del modelo

Para la generación del archivo de rutas y del modelo se usan las herramientas que proporciona ModelNet.

La primera de ellas se encarga de generar la ruta más corta entre cada par de nodos a partir del gráfico de la red, para ejecutarla hay que usar el siguiente comando:

```
allpairs archivo.graph > archivo.route
```

Esta herramienta genera un camino para cada par de nodos origen-destino. Esto quiere decir que para un experimento en el que existan 1.000 nodos, este archivo tendrá 1.000.000 de entradas. En este punto se encuentra el mayor fallo de ModelNet, ya que al no haber agregación de rutas este archivo se hará inmanejable para los core node cuando se trate de emular redes medianamente grandes.

Por último, hay que generar el archivo .model a partir del gráfico de la red y las máquinas disponibles mediante el siguiente comando:

```
mkmodel archivo.graph archivo.hosts > archivo.model
```

En este archivo se mapeará cada nodo virtual a una máquina que actuará como edge node y cada emulador a una de las máquinas que actúa como core node.

Despliegue de la red

Una vez generados los archivos con las rutas y el modelo es necesario enviarlos a todas las máquinas que formarán parte de la red ModelNet. Una vez enviados los archivos hay que ejecutar el siguiente comando en cada máquina.

```
deployhost archivo.model archivo.route
```

De esta manera se construyen las tablas de rutas en cada nodo y emulador con el fin de encaminar los paquetes de forma correcta por la red.

Ejecución de un programa

Una vez que la topología de red se ha desplegado en el hardware de emulación, el sistema está activo y se puede probar una aplicación distribuida. Todos los paquetes que un edge node mande a la red 10.0.0.0/8 serán enviados hacia su emulador.

Para poder diferenciar los paquetes que provienen de un emulador y de un nodo virtual ModelNet cambia el bit número 9 de la dirección IP al mandar un paquete en un nodo virtual. De modo que convierte las IP 10.0.X.X en 10.128.X.X. Una vez el paquete ha sido enrutado a través de los emuladores se vuelve a cambiar el noveno bit y se envía a su destino con la dirección IP original.

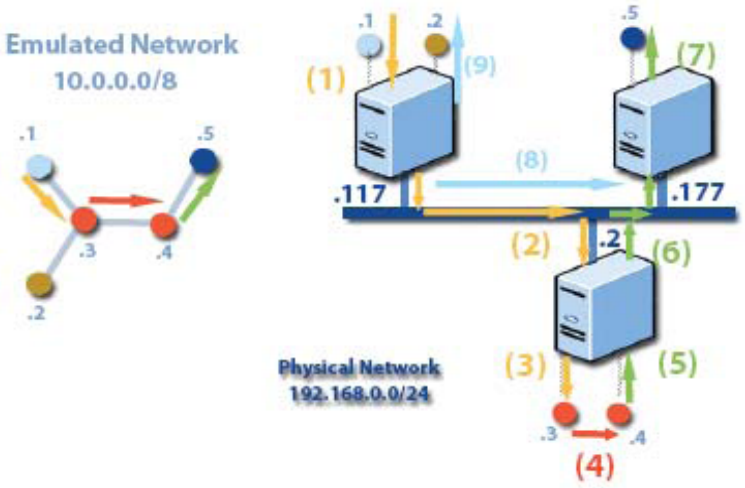


Figura B.1: Red de ejemplo en ModelNet

Para lanzar una aplicación en ModelNet hay que ejecutar el siguiente comando.

```
vnrun [-d] < VN# | all > <archivo.model> <comando>
```

Si se quiere que el comando se ejecute en todos los nodos virtuales de la máquina hay que elegir la opción “all” y si se quiere que sólo se ejecute en uno de los nodos hay que indicar el número del nodo.

C - Proxmox Virtual Environment

Este apéndice ha sido extraído del aparecido en la versión de partida [SENB07a], y que por motivos de comodidad para el lector se ha introducido también en esta versión.

Introducción

Proxmox Virtual Environment (Proxmox VE) [PROX09] es una plataforma de virtualización de código libre desarrollado y mantenido por Proxmox Server Solutions GmbH y financiado por la Internet Foundation Austria. Proxmox VE permite correr aplicaciones virtuales y máquinas virtuales de una manera muy sencilla.

La última versión estable de Proxmox VE está basada en un Debian Lenny de 64 bits. Usa una partición del tipo LVM2 para el disco duro e incluye el Kernel de Proxmox VE (la última versión es la 2.6.24), en el cuál están incluidos parches para soportar los sistemas OpenVZ y KVM.

Una de las características más importantes de Proxmox VE es que permite crear una estructura de servidores completa a partir de un hardware sin ninguna instalación previa en menos de una hora incluyendo proxys de email y web, wikis, intranets… Contando además con la ventaja de que se tiene un sistema de copias de seguridad integrado y la posibilidad de migrar las máquinas virtuales entre distintas máquinas físicas sin la necesidad de pararlas. Esta facilidad y rapidez en la instalación es debida a que Proxmox VE proporciona aplicaciones virtuales prefabricadas y una interfaz, vía web, que permite realizar la gestión de una manera muy sencilla.

Ventajas e inconvenientes de la virtualización

Entre las ventajas más importantes que se tiene al usar una plataforma de virtualización es que de este modo se puede aprovechar al máximo el hardware. En los últimos años, con la incorporación al mercado de los procesadores de 64 bits, ha aparecido la posibilidad de usar cantidades de memoria RAM muy superiores al máximo de 4Gb que marcaban los procesadores de 32 bits. Sin embargo, también existe la paradoja de que tras invertir grandes cantidades de dinero en equipos con mayor memoria y mejor procesador, muchas aplicaciones no utilizan todo este potencial al haber sido diseñadas para trabajar únicamente con direccionamiento de 32 bits. En este caso, plataformas como Proxmox (que trabaja sobre 64 bits) permiten al usuario aprovechar todos los recursos de las máquinas al permitir repartir la memoria real de la máquina física entre diferentes máquinas virtuales.

Otra ventaja importante que se obtiene al usar una plataforma de virtualización es la reducción de costes. El usar de forma más óptima el hardware permite tener menos máquinas físicas lo cual se traduce en un menor gasto de energía (permitiendo por tanto reducir el CO_2 generado, muy importante en estos momentos) y además permite tener menos personal de mantenimiento.

Dejando a un lado los beneficios económicos, también hay que destacar que con plataformas como Proxmox VE se permite ahorrar tiempo en la instalación, ya que proveen de aplicaciones prefabricadas listas para ser usadas en cuestión de minutos, ahorrándose el proceso de instalación y configuración de las mismas.

Por último, existe otra gran ventaja en Proxmox, que es la capacidad de realizar copias de seguridad y restaurarlas de forma sencilla. De esta forma se puede tener un backup del sistema de una manera muchos más sencilla que en un equipo físico.

Sin embargo, no todo son ventajas, y en la virtualización existen algunas desventajas claras respecto al trabajo directamente sobre máquinas físicas.

La desventaja más importante de usar virtualización según [DUBI08] es que magnifica los fallos de hardware. Parece obvio pensar que si se tienen 4 máquinas virtuales en la misma máquina física y ésta se desconecta o avería, se tendrán averiadas 4 máquinas en lugar de una. En un caso de fallo normal, si un servidor falla se puede tener servidores de backup que se ocupen de sus tareas, si fallan 4 servidores a la vez será más difícil que el sistema completo siga funcionando.

Tampoco hay que olvidarse que la inclusión de un sistema de virtualización gasta recursos, por lo que una parte del hardware será usado para el mantenimiento de las máquinas virtuales, perdiendo de este modo parte de la capacidad.

También es importante resaltar que del mismo modo que esta solución ofrece la posibilidad de descargar máquinas virtuales preinstaladas y configuradas, también es posible que muchas aplicaciones no funcionen sobre un sistema virtualizado y algunas no lo hagan de una forma óptima. Además, al añadir una nueva capa de complejidad al sistema es mucho más difícil el encontrar donde se producen los fallos del sistema.

Por último cabe destacar, que partes de una máquina física como la tarjeta de red o la velocidad del disco duro, que generalmente no limitan al sistema, pero que, si la aplicación que corre en el nivel superior es muy exigente con ellas puede que en este caso si sean limitantes. No hay que olvidar que ahora se tiene sólo un disco duro (o raid) y una tarjeta de red para varias máquinas virtuales.

Instalación

La instalación del sistema Proxmox VE sobre una máquina física no difiere demasiado de la instalación de una distribución Debian normal. Tras arrancar el sistema desde el CD de instalación se deben seguir sus instrucciones. La instalación requerirá la atención del usuario en varias ocasiones a lo largo de su desarrollo. En un primer lugar pedirá que se acepte la licencia y el disco duro en el que Proxmox VE quiere instalarse. Posteriormente se dará al sistema información sobre la zona geográfica, pedirá la clave y el mail de administrador y después los datos de configuración de la red. A partir de este momento, la instalación ya no requerirá más atención, y tras un breve periodo el sistema se encontrará instalado.

Una vez finalizada la copia de archivos al equipo se debe iniciar la configuración del sistema. Para ello, hay que acceder al interfaz mediante un navegador web, en la actualidad Proxmox soporta Microsoft Internet Explorer a partir de la versión 6, y Mozilla Firefox a partir de la versión 2. Una vez aquí, aparecerá la siguiente interfaz.

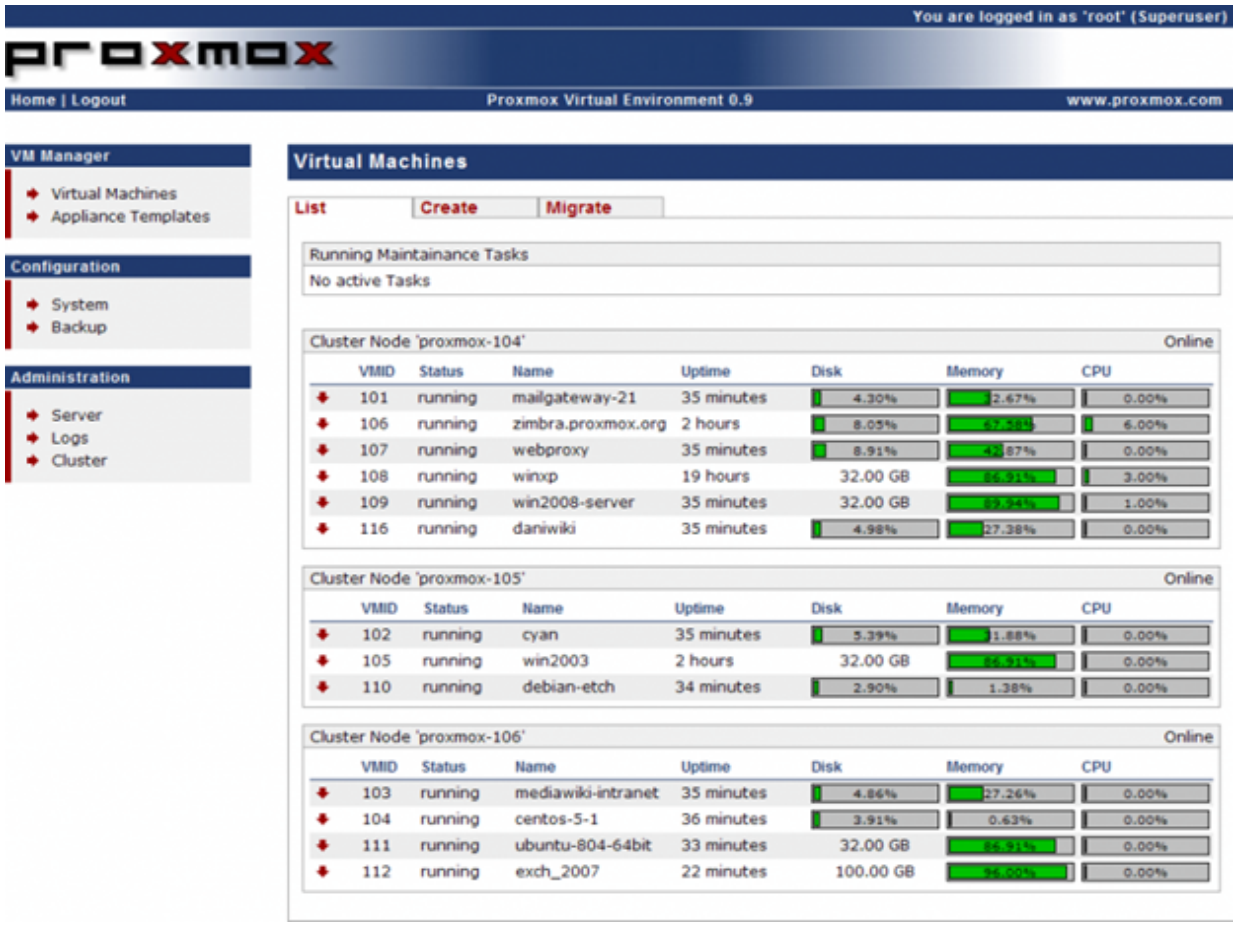


Figura C.1: Captura de pantalla de Proxmox (1)

Y dentro del apartado System se podrá configurar las interfaces de red del sistema o el servidor DNS a utilizar.

Configuración del cluster

Proxmox VE permite tener varias máquinas físicas unidas dentro de un cluster, para de ese modo, poder manejar todas las máquinas físicas desde una sola, que será la máquina principal.

Para añadir una máquina a un cluster se debe ejecutar el siguiente comando desde la máquina que se desea añadir:

```
pveca -a -h IP-ADDRESS-MASTER
```

Creación de las máquinas virtuales

Una vez instalado el sistema Proxmox VE, se podrán empezar a crear las máquinas virtuales.

El sistema Proxmox VE proporciona dos tipos de máquinas virtuales, los OpenVZ y los KVM cuyo comportamiento es diferente en el sistema. Para la creación de ambos tipos hay que irse al apartado "Create" dentro del apartado "Virtual Machines".

Creando un OpenVZ

Los OpenVZ son contenedores en los que se instala una plantilla prefabricada de las que se pueden descargar en el apartado "Virtual Templates" o que se pueden subir desde la propia máquina. Para poder acceder a estas plantillas hay que seleccionar como tipo de máquina virtual el tipo "Container (OpenVZ)". Se verán los siguientes apartados de configuración:

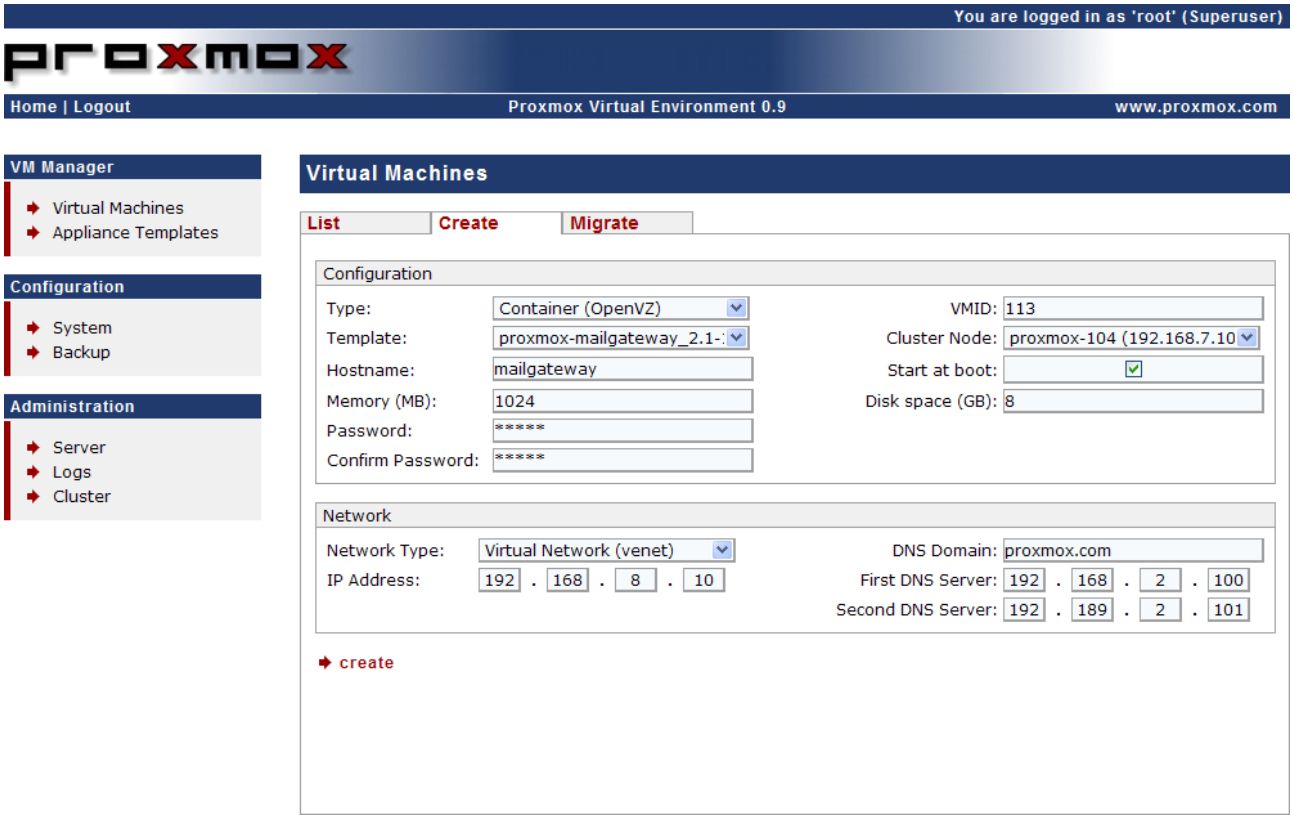


Figura C.2: Captura de pantalla Proxmox (2)

- Type: Seleccionar "Container (OpenVZ)".

- Template: en este apartado se tienen disponibles todas las plantillas que se hayan añadido previamente.

- Hostname: un nombre único para esta máquina virtual.

- Memory (MB): memoria RAM asignada a esta máquina virtual.

- Swap (MB): memoria Swap asignada a esta máquina virtual. En la versión actual de Proxmox esta cantidad de memoria es sumada directamente a la memoria RAM.

- Password: la contraseña de administrador.

- VMID: el identificador de la máquina virtual. Este debe ser único, y si se cambia por uno ya existente se sobrescribirá la máquina virtual con el otro identificador.

- Cluster Node: en una lista se ven los nodos disponibles y se puede elegir en que nodo se quiere crear la máquina virtual.

- Start at boot: si se activa este apartado, la máquina virtual se encenderá automáticamente cuando se inicie el sistema.

- Disk Space (GB): Marca el espacio máximo que puede ocupar en disco la máquina virtual. Sin embargo, en el caso de los OpenVZ este espacio no es reservado de ante mano en el disco duro físico, sino que se va llenando según se necesita.

Por último se puede configurar el apartado de red (“Network”) asignando a la máquina virtual una o varias interfaces de red que previamente se habrán creado en el sistema.

Creando un KVM

Los KVM son máquinas completamente virtualizadas. Al contrario que en el caso de los OpenVZ ahora se debe realizar una instalación y configuración completa del sistema instalado. En este caso el tipo a elegir será "Fully virtualized (KVM)". La creación es muy similar al caso de un OpenVZ por lo que se pasan a describir lo apartados diferentes.

- Type: elegir “Fully virtualized (KVM)”.

- Installation Media: aquí se puede elegir “cd rom” si se quiere realizar la instalación desde la unidad de CD física el equipo, o una imagen de CD que previamente se haya subido al sistema.

- Disk Space (GB): En este caso el espacio en disco elegido es reservado completamente al crear la máquina virtual.

- Disk type: seleccionar el tipo de disco duro deseado.

- Guest Type: elegir el tipo de sistema que se va a instalar (32 o 64 bits están permitidos).

Una vez creada la máquina virtual, arrancarla y proceder a instalar el sistema deseado.

D - **Países participantes en la prueba**

Todos los datos que aquí se muestran están obtenidos de la página de la CIA [CIA10b] y de una página que recoge las estadísticas de los usuarios de Internet por países [IWS10]. Por último, comentar que los valores que aparecen en las tablas del número de peers se han redondeado para que siempre el numero de peers por experimento sea de 1000.

1 Dominio

Nº	PAÍS	USUAR INT	PORCENTAJE	PEERS	PENETRACIÓN	FECHA
1	CHINA	346000000	100	1000	28,7	Dic-2009

5 Dominio

Nº	PAÍS	USUAR INT	PORCENTAJE	PEERS	PENETRACIÓN	FECHA
1	CHINA	346000000	69,56	695,58	28,7	Dic-2009
2	FRANCIA	44625300	8,97	89,71	68,9	Jun-2010
3	ITALIA	30026400	6,04	60,36	51,7	Ago-2009
4	DINAMARCA	4750500	0,96	9,55	86,1	Jun-2010
5	BRASIL	72027700	14,48	144,8	36,2	Dic-2009

10 Dominios

Nº	PAÍS	USUAR INT	PORCENTAJE	PEERS	PENETRACIÓN	FECHA
1	CHINA	346000000	39,93	399,29	28,7	Dic-2009
2	FRANCIA	44625300	5,15	51,5	68,9	Jun-2010
3	ITALIA	30026400	3,47	34,65	51,7	Ago-2009
4	DINAMARCA	4750500	0,55	5,48	86,1	Jun-2010
5	BRASIL	72027700	8,31	83,12	36,2	Dic-2009
6	ALEMANIA	62123800	7,17	71,69	79,1	Jun-2010
7	EEUU	234372000	27,05	270,47	76,3	Nov-2009
8	TAIWAN	15143000	1,75	17,48	65,9	Jun-2009
9	COREA S	37475800	4,32	43,25	77,3	Jun-2009
10	ARGENTINA	20000000	2,31	23,08	49,4	Dic-2008

20 Dominios

Nº	PAÍS	USUAR INT	PORCENTAJE	PEERS	PENETRACIÓN	FECHA
1	CHINA	346000000	28,29	282,87	28,7	Dic-2009
2	FRANCIA	44625300	3,65	36,48	68,9	Jun-2010
3	ITALIA	30026400	2,45	24,55	51,7	Ago-2009
4	DINAMARCA	4750500	0,39	3,88	86,1	Jun-2010
5	BRASIL	72027700	5,89	58,89	36,2	Dic-2009
6	ALEMANIA	65123800	5,32	53,24	79,1	Jun-2010
7	EEUU	234372000	19,16	191,61	76,3	Nov-2009
8	TAIWAN	15143000	1,24	12,38	65,9	Jun-2009
9	COREA S	37475800	3,06	30,64	77,3	Jun-2009
10	ARGENTINA	20000000	1,64	16,35	49,4	Dic-2008
11	PORTUGAL	5168800	0,42	4,23	48,1	Jun-2010
12	REINO UNIDO	51442100	4,21	42,06	82,5	Jun-2010
13	JAPON	95979000	7,85	78,47	75,5	Sep-2009
14	INDIA	81000000	6,62	66,22	7	Nov-2008
15	TAILANDIA	16100000	1,32	13,16	24,4	Sep-2009
16	MEJICO	30600000	2,5	25,02	27,2	Dic-2009
17	SUIZA	5739300	0,47	4,69	75,5	Sep-2009
18	SUDAFRICA	5300000	0,43	4,33	10,8	Dic-2009
19	RUSIA	45250000	3,7	36,99	32,3	2009
20	AUSTRALIA	17033826	1,39	13,93	80,1	2009

30 Dominios

Nº	PAÍS	USUAR INT	PORCENTAJE	PEERS	PENETRACIÓN	FECHA
1	CHINA	346000000	24,53	245,32	28,7	Dic-2009
2	FRANCIA	44625300	3,16	31,64	68,9	Jun-2010
3	ITALIA	30026400	2,13	21,29	51,7	Ago-2009
4	DINAMARCA	4750500	0,34	3,37	86,1	Jun-2010
5	BRASIL	72027700	5,11	51,07	36,2	Dic-2009
6	ALEMANIA	65123800	4,62	46,17	79,1	Jun-2010
7	EEUU	234372000	16,62	166,17	76,3	Nov-2009
8	TAIWAN	15143000	1,07	10,74	65,9	Jun-2009
9	COREA S	37475800	2,66	26,57	77,3	Jun-2009
10	ARGENTINA	20000000	1,42	14,18	49,4	Dic-2008
11	PORTUGAL	5168800	0,37	3,66	48,1	Jun-2010
12	REINO UNIDO	51442100	3,65	36,47	82,5	Jun-2010
13	JAPON	95979000	6,81	68,05	75,5	Sep-2009
14	INDIA	81000000	5,74	57,43	7	Nov-2008
15	TAILANDIA	16100000	1,14	11,42	24,4	Sep-2009
16	MEJICO	30600000	2,17	21,7	27,2	Dic-2009
17	SUIZA	5739300	0,41	4,07	75,5	Sep-2009
18	SUDAFRICA	5300000	0,38	3,76	10,8	Dic-2009
19	RUSIA	45250000	3,21	32,08	32,3	2009
20	AUSTRALIA	17033826	1,21	12,08	80,1	2009
21	CANADA	25086000	1,78	17,79	74,9	Sep-2009
22	ARABIA SAUDI	9800000	0,69	6,95	38,1	2009
23	TURQUIA	26500000	1,88	18,79	34,5	Mar-2008
24	INDONESIA	30000000	2,13	21,27	12,5	Sep-2009
25	PAISES BAJOS	14872200	1,05	10,54	88,6	Jun-2010
26	POLONIA	22450600	1,59	15,92	58,4	Jun-2010
27	BELGICA	8113200	0,58	5,75	77,8	Jun-2010
28	VENEZUELA	8846535	0,63	6,27	33	Dic-2009
29	IRAN	33200000	2,35	23,54	43,2	Jun-2010
30	CHILE	8369036	0,59	5,93	50,4	Dic-2008

40 Dominios

Nº	PAÍS	USUAR INT	PORCENTAJE	PEERS	PENETRACIÓN	FECHA
1	CHINA	346000000	22,98	229,8	28,7	Dic-2009
2	FRANCIA	44625300	2,96	29,64	68,9	Jun-2010
3	ITALIA	30026400	1,99	19,94	51,7	Ago-2009
4	DINAMARCA	4750500	0,32	3,16	86,1	Jun-2010
5	BRASIL	72027700	4,78	47,84	36,2	Dic-2009
6	ALEMANIA	65123800	4,33	43,25	79,1	Jun-2010
7	EEUU	234372000	15,57	155,66	76,3	Nov-2009
8	TAIWAN	15143000	1,01	10,06	65,9	Jun-2009
9	COREA S	37475800	2,49	24,89	77,3	Jun-2009
10	ARGENTINA	20000000	1,33	13,28	49,4	Dic-2008
11	PORTUGAL	5168800	0,34	3,43	48,1	Jun-2010
12	REINO UNIDO	51442100	3,42	34,17	82,5	Jun-2010
13	JAPON	95979000	6,37	63,75	75,5	Sep-2009
14	INDIA	81000000	5,38	53,8	7	Nov-2008
15	TAILANDIA	16100000	1,07	10,69	24,4	Sep-2009
16	MEJICO	30600000	2,03	20,32	27,2	Dic-2009
17	SUIZA	5739300	0,38	3,81	75,5	Sep-2009
18	SUDAFRICA	5300000	0,35	3,52	10,8	Dic-2009
19	RUSIA	45250000	3,01	30,05	32,3	2009
20	AUSTRALIA	17033826	1,13	11,31	80,1	2009
21	CANADA	25086000	1,67	16,66	74,9	Sep-2009
22	ARABIA SAUDI	9800000	0,65	6,51	38,1	2009
23	TURQUIA	26500000	1,76	17,6	34,5	Mar-2008
24	INDONESIA	30000000	1,99	19,93	12,5	Sep-2009
25	PAISES BAJOS	14872200	0,99	9,88	88,6	Jun-2010
26	POLONIA	22450600	1,49	14,91	58,4	Jun-2010
27	BELGICA	8113200	0,54	5,39	77,8	Jun-2010
28	VENEZUELA	8846535	0,59	5,88	33	Dic-2009
29	IRAN	33200000	2,21	22,05	43,2	Jun-2010
30	CHILE	8369036	0,56	5,56	50,4	Dic-2008
31	COLOMBIA	20788818	1,38	13,81	47,6	Sep-2009
32	CUBA	1604000	0,11	1,07	14	Jun-2010
33	ECUADOR	1840678	0,12	1,22	12,6	Sep-2009
34	EGIPTO	16660000	1,11	11,07	21,1	Dic-2009
35	GUATEMALA	2280000	0,15	1,51	16,8	Jun-2010
36	NIGERIA	23982200	1,59	15,93	16,1	Dic-2009
37	PAKISTAN	18500000	1,23	12,29	10,6	Jun-2009
38	PARAGUAY	894200	0,06	0,59	12,8	Sep-2009
39	PERU	7636400	0,51	5,07	25,8	Mar-2008
40	BOLIVIA	1050000	0,07	0,7	10,7	Dic-2009

REFERENCIAS Y BIBLIOGRAFÍA

Referencias

[API10] http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones, Accedido en Marzo 2010.

[BYT10] http://es.wikipedia.org/wiki/Bytecode, Accedido en Abril 2010.

[CAID10] http://www.caida.org/. Proyecto CAIDA. Accedido en Abril 2010.

[CIA10a] https://www.cia.gov/.

[CIA10b] https://www.cia.gov/library/publications/the-world-factbook/rankorder/2153rank.html

[CRI10] http://xml.apache.org/crimson, Accedido en Abril 2010

[DIV10] http://es.wikipedia.org/wiki/DivX, Accedido en Abril 2010.

[DOM10] http://www.w3.org/DOM/, Accedido en Marzo 2010

[DTD10] http://www.w3.org/TR/xhtml1/dtds.html. Accedido en Marzo 2010.

[DUBI08] Denise Dubie. 7 side effects of sloppy virtualization. En Nework World. 2008 http://www.networkworld.com/news/2008/062408-sloppy-virtualization.html?page=1

[DUM10] http://info.iet.unipi.it/~luigi/dummynet/

[EDO10] http://en.wikipedia.org/wiki/EDonkey_network, Accedido en Abril 2010

[GNU10] http://es.wikipedia.org/wiki/Gnutella, Accedido en Abril 2010

[IEE10] http://www.ieee.org/index.html, IEEE Standard Computer Dictionary: A Compilation IEEE Standard Computer Glossaries,New York: IEEE,1990. Accedido en Abril 2010

[IETF10] http://www.ietf.org Accedido en Diciembre 2010. Internet Engineering Task Force

[IWS10] http://www.internetworldstats.com

[JAM10] http://es.wikipedia.org/wiki/James_Gosling, Accedido en Abril 2010

[JAV10] http://www.java.com, Accedido en Abril 2010

[JAX10] https://jaxp.dev.java.net/, Accedido en Abril 2010

[JDOM10] http://www.jdom.org/, Accedido en Abril 2010

[JXT10] https://jxta.dev.java.net/, Accedido en Abril 2010.

[KAD10] http://e-archivo.uc3m.es/handle/10016/8382 Accedido en Abril de 2010.

[KAZ10] http://www.kazaa.com, Accedido en Abril 2010

[MAT10] http://www.mathworks.com/

[MIC10] http://www.microsoft.com, Accedido en Abril 2010

[MOD10] https://modelnet.sysnet.ucsd.edu/, Accedido en Mayo 2010

[MM02] P. Maymounkov and D. Mazieres. Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7-8, 2002. Revised Papers, volume 2429/2002 of Lecture Notes in Computer Science, chapter Kademlia: A peer-to-peer information system based on the XOR metric, páginas 53–65. Springer, 2002.

[MP310] http://es.wikipedia.org/wiki/MP3, Accedido en Abril 2010.

[MSN10] http://windowslive.es.msn.com/messenger/, Accedido en Abril 2010.

[MYBC+09] Isaías Martínez-Yelmo, Alex Bikfalvi, Rubén Cuevas, Carmen Guerrero, y Jaime García. *H-p2psip: Interconnection of p2psip domains for global multimedia services based on a hierarchical dht overlay network*. Computer Networks (Special -Issue on Content Distribution Infrastructures for Community Networks), 53(4), Mar. 2009.

[NAP10] http://www.napster.com/, Accedido en Abril 2010

[NET10] http://netbeans.org/. Accedido en Marzo de 2010.

[NIS10] http://www-x.antd.nist.gov/nistnet/

[P2PP09] http://www1.cs.columbia.edu/~salman/peer/. Peer to Peer Protocol. 2009.

[PIN10] http://www.iepm.slac.stanford.edu/pinger/

[PLA10] http://www.planet-lab.org

[PXP10] http://www.jclark.com/xml/xp/, Accedido en Abril 2010
[SAX10] http://www.saxproject.org , Accedido en Marzo 2010
[SENB07a] Roberto González Sánchez. Implementación de redes overlay jerárquicas usando el protocolo P2PP.
[SENB07b] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. Analyzing peer behavior in KAD. Technical Report EURECOM+2358, Institut Eurecom, France, Oct 2007.
[SENB07c] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A global view of kad. In IMC ´07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, páginas 117-122, New York, NY, USA, 2007. ACM. France, Oct 2007.
[SGM10] http://www.w3.org/MarkUp/SGML/. Accedido en Marzo de 2010
[SIP10] http://es.wikipedia.org/wiki/Session_Initiation_Protocol/ . Accedido en Dic de 2010
[SKY10] http://www.skype.com/intl/es/home/, Accedido en Abril 2010.
[SUN10] http://es.wikipedia.org/wiki/Sun_Microsystems, Accedido en Abril 2010.
[VBO10] http://www.virtualbox.org/
[VOIP10] http://es.wikipedia.org/wiki/Voz_sobre_IP . Accedido en Diciembre de 2010
[W3C10] http://www.w3.org/. Accedido en Marzo de 2010
[WAN10] http://wanem.sourceforge.net/
[WMW10] http://www.vmware.com/
[XER10] http://xerces.apache.org/xerces-j/, Accedido en Abril 2010
[XMD10] http://www.w3schools.com/Dom/dom_parser.asp, Accedido en Abril 2010
[XML10] http://www.w3.org/XML/. Accedido en Marzo de 2010
[XSL10] http://www.w3.org/TR/xslt, Accedido en Marzo 2010
[YMS10] http://es.messenger.yahoo.com/, Accedido en Abril 2010.

Bibliografía

- Martín, G., and Martín I. : Curso de XML (Pearson Prentice Hall, 2005, 1ª Edición)

- McLaughlin B., and Edelson J. :Java & XML (O´Reilly, 2007, 3ª Edición)

- Eckel B. :Thinking in Java (Prentice Hall, 2000, 2ª Edición)

- Millán R. : Domine las redes P2P (Creaciones Copyright, 2006, 1º Edición)

- Buford F. , and Yu H. , Keong E. :P2P Networking and Applications (Elsevier ,2009)

GLOSARIO

Bootstrap

Servicio que permite el arranque del sistema. En este caso, es un elemento que actúa como servidor de bootstrap y será el encargado de proporcionar a los nuevos peers información sobre los nodos que ya forman parte de la red.

Chord

Protocolo de búsqueda distribuida. Es el más sencillo de todos los que describen una red overlay estructurada.

Cluster

Grupo de nodos (también denominado dominio).

DHCP

Dynamic Host Configuration Protocol. Protocolo de red que permite a los dispositivos participantes en una red IP obtener sus parámetros de configuración automáticamente.

DHT

Distributed hash table. Describe una clase de servicios descentralizados que proporcionan una capacidad de búsqueda parecida a la de una tabla de hash.

DNS

Domain Name System. Sistema de nomenclatura para ordenadores, servicios o cualquier recurso conectado a Internet o a una red privada de forma jerárquica.

Gateway

Dispositivo que permite interconectar redes con diferentes protocolos o arquitecturas.

Hash

Una función de hash proporciona una clave de manera casi unívoca que representa al contenido deseado. De ese modo se pueden tener claves de un tamaño prefijado.

H-P2PSIP

Hierarchical Peer-to-peer SIP. Es una versión jerárquica del protocolo P2PSIP.

ID

Identificador.

IP

Etiqueta numérica que identifica, de manera lógica y jerárquica a un interfaz de un dispositivo dentro de una red que utilice el protocolo IP (Internet Protocol) que corresponde al nivel de red del protocolo TCP/IP.

IETF

Internet Engineering Task Force. Es una comunidad internacional de diseñadores, operadores, comerciantes de red e investigadores que se centra en la evolución de la arquitectura de Internet.

Kademlia

Protocolo de búsqueda distribuida. Se trata de un protocolo que describe una red overlay estructurada.

Key

Cada una de las posibles claves que representan a un peer o a algún tipo de contenido dentro de la red.

MAC

Media Access Control. Una dirección MAC es un identificador de 48 bits que corresponde de forma univoca a una Ethernet de red.

ModelNet

Emulador de red usado en este proyecto.

Napster

Primera aplicación P2P que apareció. Desarrollada en 1996 pretendía ser una plataforma de distribución de archivos.

NAT

Network Address Translation. Es un mecanismo utilizado por routers IP que permite la comunicación de redes que tienen direcciones incompatibles. En la práctica se usa de manera generalizada para la traducción de direcciones privadas en públicas.

Node

Cada uno de los elementos participantes en una red overlay.

P2PSIP

Peer-to-peer SIP es un protocolo que permite la creación de redes overlay. Actualmente está siendo estandarizado por el IETF.

Peer

Cada uno de los elementos en una red overlay y que participa en el encaminamiento de mensajes.

PlanetLAB

Gran laboratorio computacional albergado dentro de las redes académicas del mundo, cuyo objetivo es poner a disposición de la comunidad académica recursos para probar aplicaciones que requieran una red de datos para su ejecución.

Proxmox

Plataforma de virtualización de código libre que permite correr aplicaciones y máquinas virtuales de una manera muy sencilla.

Query

Búsqueda en una red overlay.

RELOAD

REsource LOcation And Discovery. Es el protocolo de señalización usado por P2PSIP.

Rtt

Round-Trip delay Time. Tiempo que tarda un paquete enviado por un emisor en volver a este mismo tras pasar por el receptor de destino.

Skype

Software usado para realizar llamadas de VoIP. Es de código cerrado. Utiliza una red overlay para comunicar a los diferentes clientes.

URI

Uniform Resource Identifier. Identificador uniforme de recurso.

VoIP

Voz sobre IP. Grupo de recursos que hacen posible que la voz viaje a través de internet encapsulada en paquetes IP.

WG

Working group. Grupo de trabajo del IETF.

NOTAS